# Embedded Systems in Distributed Environment

*Second Workshop on Distributed Laboratory Instrumentation Systems*

Abdus Salam ICTP, Trieste, 20 October – 14 November 2003

Chu Suan ANG
Kuala Lumpur
Malaysia

*Email: csang@pc.jaring.my*

**Abstract**

Several embedded system core components, ranging from small single chip microcontrollers to large CPU modules, are described in the first part of these lectures. Embedded systems networking in a distributed environment is covered in the second part with a bias towards techniques used in the laboratory environment, taking into account both legacy and new systems.

# 1 Embedded System Processors and Modules

## 1.1 Introduction

The overall objectives of these lectures are (1) to introduce the current embedded system hardware components or subsystems that are both available and affordable, and (2) to introduce the rudiments of networking various small embedded systems including both new and old (legacy) components.

The scene of embedded processors has changed significantly. Motorola, Intel and Zilog are no longer the obvious choices for most small research or teaching laboratories. Instead, Dallas Semiconductor, Atmel, Microchip, Rabbit Semiconductor and others are getting more and more competitive and have been widely adopted in laboratories. Several types of embedded processors suitable for embedded systems, spanning a wide range of capabilities and thus costs, are introduced here.

It is of course impossible to deal exhaustively with the available processors or subsystems in the market in a few lectures. One may tend to group them and do a generalized treatment. But a general idea of hardware is not sufficient for building an embedded system. At the end of the Workshop, it is hoped that participants will acquire sufficient knowledge to design and build their own systems. General treatment will not achieve that. The devil is in the detail.

Hardware or hardware subsystems for five different processors, ranging from the very small and simple to the complex but affordable, are chosen as examples of processors in the different embedded system classes.

1. PIC
2. TINI
3. Rabbit Core
4. Etrax MCM
5. ICOP PC/104 Embedded CPU Module

The PIC family of microcontrollers by Microchip has been extremely successful for small embedded system applications. The smaller members of PIC are under one dollar which is truly low cost. Of course a tiny eight-legged MCU is not quite a full embedded system but you will be surprised how little more is required to make it so. With an additional few dollars one can have a small PCB, connectors and a small line driver to make it a fully-fledged embedded system, which can be networked together via a serial bus such as the RS485. It will count events, it will control LEDs, it will also sense switches, and it will even read analogue voltages. Many of them can be distributed over a large area and connected to a PC or bigger system using a serial bus which is just a twisted pair of wires. This makes distributed systems in laboratories really feasible at low cost. A PIC system as small as a match box costing under 5 dollars has really more or less the capabilities of the Motorola board used in the early microprocessor colleges that preceded this Workshop in the 1980s.

TINI is the workhorse in this Workshop. It is one of the rising stars in the not-so-small embedded microcontrollers and it falls in the category of 50-100 dollar systems. The processor together with its many virtues is dealt with separately in another series of lectures and shall not be covered here.

The Rabbit Cores by Rabbit Semiconductor fall in the same category as the TINI. With an early successful entry in the market, there are now many different configurations and accessories available to suit different requirements. With an on-board Ethernet connector on the core module, the extension board can usually be quite simple, and therefore possible for participants to produce in their home. It is also a good candidate to use for interfacing to legacy serial equipment. It is of course a rather powerful embedded system in its own right and can be used in many localized tasks that require hard real-time control, using a real-time kernel such as uCOS. A software development environment under the Linux platform is being carried out by two members of the Workshop.

It was realized several years ago that a complete system, including memories and peripherals on a single chip with processing power equivalent to a low end PC, was feasible. Well, one such processor is now available. It is the ETRRAX 100LX Multi Chip Module (MCM) by Axis Communications. At 70 dollars, the chip has the processing power and hardware resources to run Linux and other OS of similar complexity. It is a good candidate for complex real-time systems in a distributed environment in view of its high level OS capabilities and low cost.

It is not true that one has to go for the cheapest possible hardware all the time. Many a time it is wiser to acquire appropriate hardware to do the job irrespective of the cost. Off-the-shelf subsystems are often more cost effective in the end because of the time saved in using them. In embedded systems, one such group is the commercially available high end embedded CPU modules. They are, in general, entire PCs in cards called single board computers (SBC). Many standards and form factors have appeared over the years, including the STB Bus, PC/104, EBX, half-size SBC, and 3.5” embedded module. The large number of manufacturers each producing a myriad of boards makes the choice of such boards difficult. PC/104 is a small form-factor (3.6 x 3.8 inches, 90 x 96 mm) worldwide standard now gaining popularity. It supports both ISA and PCI expansion in self-stacking modules, at 0.6 inch spacing. One such card is the Vortex86-6070 by ICOP Technology.

If you compare the above board with that of Axis 82 using Etrax 100LX MCM, you will see that it is definitely more suitable for bigger jobs. While the raw processing power of the two is comparable, the Vortex86 has significantly larger on-board memory. Expansion is straightforward with PC/104 modules. A display controller is also embedded, which may be required in some embedded applications (e.g. mimic diagram or synoptic panel of the railway network on a high resolution colour LCD panel?).

Basically such an embedded processor is more or less like a desktop PC in terms of computing power and peripherals such as mass storage and display. The programming environment is practically identical to that of a desktop PC and cross development becomes straightforward or perhaps unnecessary for the system itself can be a development environment. The differences lie in its compactness and economy – the whole embedded system can be held in a palm and probably costs half that of the desktop PC doing the same job.

## 1.2 PICmicro® MCUs

PICmicro® MCUs are RISC-based devices produced by Microchip Technology Inc. that have had tremendous success in replacing older microcontrollers in small embedded system applications in recent years. The family of MCUs has more than 140 products featuring a variety of memory configurations, low voltage and power, small footprints and ease-of-use. These products can be grouped into five families of 8-bit MCUs:

**PIC12CXXX/PIC12FXXX Family: 8-pin 12-bit/14-bit program word**
These are the smallest MCUs using RISC-based PICmicro® architecture housed in 8-pin DIP and SOIC packages. They are available with either a 12-bit or 14-bit wide instruction set, a low operating voltage of 2.5V, small package footprints, interrupt handling, a deeper hardware stack, multiple A/D channels, FLASH, OTP or ROM program memory, and EEPROM data memory.

**PIC16C5X Family: 12-bit program word**
These are very popular MCUs because they offer cost-effective solutions in many embedded system applications. They have a 12-bit wide instruction sets and are currently offered in 14-, 18-, 20- and 28-pin packages, in the SOIC and SSOP packaging options. Low-voltage operation, down to 2.0V for OTP MCUs, makes this family ideal for battery-operated applications. The PIC16HV5XX can operate up to 15 volts for use directly with a 12-volt battery.

**PIC16CXXX/PIC16FXXX Family: 14-bit program word**
A new family that provides high performance Analog-to-Digital Converter capability at 12-bits for an MCU. The family offers a wide-range of options, from 18- to 68-pin packages as well as low to high levels of peripheral integration. This family has a 14-bit wide instruction set, interrupt handling capability and a deep, 8-level hardware stack.

**PIC17CXXX Family: 16-bit program word**
This family has a 16-bit instruction word, enhanced instruction set and powerful vectored interrupt-handling capabilities. A powerful array of precise on-chip peripheral features provides the performance for more demanding applications.

**PIC18CXXX/PIC18FXXX Family: enhanced 16-bit program word**
Top of the line of the PIC devices. The family has enhanced core features, ADC, 32 level-deep stacks, and multiple internal and external interrupts sources. Other nice features include programmable Low Voltage Detect (LVD) and programmable Brown-Out Detect (BOD). The separate instruction and data busses of the Harvard architecture allow a 16-bit wide instruction word with the separate 8-bit wide data. The two-stage instruction pipeline allows all instructions to execute in a single cycle, except for program branches, which require two cycles. A total of 77 instructions (reduced instruction set) are available. The MCU operates up to 10 MIPS.

## 1.3 An Example of PICmicro® MCUs PIC12F675

This is a CMOS Flash-based 8-bit microcontroller using the Microchip PIC architecture housed in an 8-pin package and featuring 4 channels of 10-bit Analog-to-Digital (A/D) converter, 1 channel Comparator, and 128 bytes of EEPROM data memory. It is used for small applications, especially when field re-programming is necessary.

The RISC CPU has only 35 instructions and they execute in one cycle except branches which require two. The clock or oscillator operates from DC to 20 MHz while the instruction cycle is DC to 200 ns. An 8-level deep hardware stack is available. Addressing modes are Direct, Indirect, and Relative.

Other secondary hardware related features are:

- Internal and external oscillator options
- External Oscillator support for crystals and resonators
- 5 µs wake-up from SLEEP
- Power saving SLEEP mode
- Wide operating voltage range: 2.0V to 5.5V

  - Industrial and Extended temperature range
  - Low power Power-on Reset (POR)
  - Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
  - Brown-out Detect (BOD)
- Watchdog Timer (WDT) with independent oscillator
  - Multiplexed MCLR/Input-pin
  - Interrupt-on-pin change
  - Individual programmable weak pull-ups
  - Programmable code protection
  - FLASH/EEPROM Cell
    - 100,000 write FLASH endurance
    - 1,000,000 write EEPROM endurance
    - FLASH/Data EEPROM Retention: > 40 years
  - Standby Current:
    - 1 nA @ 2.0V, typical
  - Operating Current:
    - 8.5µA @ 32 kHz, 2.0V, typical
    - 100µA @ 1 MHz, 2.0V, typical
  - Watchdog Timer Current
    - 300 nA @ 2.0V, typical
  - Timer1 oscillator current:
    - 4 µA @ 32 kHz, 2.0V, typical
  - Peripheral Features:
    - 6 I/O pins with individual direction control
    - High current sink/source for direct LED drive
  - Analog comparator module with:
    - One analog comparator
    - Programmable on-chip comparator voltage reference (CVREF)
    - Programmable input multiplexing from device inputs
    - Comparator output is externally accessible
  - Analog-to-Digital Converter module:
    - 10-bit resolution
    - Programmable 4-channel input
    - Voltage reference input
  - Timer0: 8-bit timer/counter with 8-bit programmable prescalar
  - Enhanced Timer1:
    - 16-bit timer/counter with prescaler
    - External Gate Input mode
  - In-Circuit Serial ProgrammingTM (ICSPTM) via two pins

## 1.4 PIC12F675 Memories and Registers

In PIC, memories are divided into program and data sections. PIC12F675 has a 13-bit program counter capable of addressing an 8K x 14 program memory space. Only the first 1K x 14 (0000h - 03FFh) is physically implemented. Accessing a location above these boundaries will cause a wrap around within the first 1K x 14 space. The RESET vector is at 0000h and the interrupt vector is at 0004h.

The data memory is partitioned into two banks, which contain the General Purpose registers and the Special Function registers. The Special Function registers are located in the first 32 locations of each bank. Register locations 20h-5Fh are General Purpose registers, implemented as static RAM and mapped across both banks. All other RAM is unimplemented and returns ‘0’ when read. RP0 is the bank select bit. RP0 = 0 selects Bank 0; RP0 = 1 selects Bank 1.

The register file is organized as 64 x 8 in the PIC12F629/675 devices. Each register is accessed, either directly or indirectly, through the File Select Register FSR.

The Special Function registers are static RAM used by the CPU and peripheral functions for controlling the desired operation of the device. Details of the core and peripheral assignment of these registers can be found in the data sheet.

The STATUS register contains the arithmetic status of the ALU, the RESET status, and the bank select bits for data memory. The STATUS register can be the destination for any instruction, like any other register. However, if the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to the device logic.

The OPTION register is a read/write register that contains various control bits to configure TMR0/WDT prescaler, external GP2/INT interrupt, TMR0, and weak pull-ups on GPIO.

The INTCON register is a read/write register that contains the various enable and flag bits for TMR0 register overflow, GPIO port change and external GP2/INT pin interrupts.

The PIE1 register contains the peripheral interrupt enable bits. The PIR1 register contains the interrupt flag bits. The Power Control (PCON) register contains flag bits to differentiate between a:

- Power-on Reset (POR)

- Brown-out Detect (BOD)
- Watchdog Timer Reset (WDT)
- External MCLR Reset

The program counter (PC) is 13-bits wide. The low byte comes from the PCL register, which is a read/write register. The high bits (PC<12:8>) are not directly accessible and they come from PCLATH. On any RESET, the PC is cleared.

The stack is 8 levels deep and 13 bits wide. It is not part of either program or data space and the stack pointer is not directly accessible. The PC is PUSHed onto the stack when a CALL instruction is executed, or an interrupt causes a branch. The stack is POPed in the event of a RETURN, RETLW or a RETFIE instruction execution. PCLATH is not affected by a PUSH or POP operation. The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. There are no status bits to indicate stack overflow or underflow. There are no instructions/mnemonics called PUSH or POP. These are actions that occur from the execution of the `CALL, RETURN,RETLW` and `RETFIE` instructions, or the vectoring to an interrupt address.

Indirect Addressing is achieved by addressing the INDF register which is not a physical register. Any instruction using the INDF register will access data pointed to by the File Select register (FSR). Reading INDF itself indirectly will produce 00h. Writing to the INDF register indirectly results in a no operation (although STATUS bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>).

## 1.5 PIC12F675 I/O Port

There are six general purpose I/O pins available. Depending on which peripherals are enabled, some or all of the pins may not be available as general purpose I/O. In general, when a peripheral is enabled, the associated pin may not be used as a general purpose I/O pin.

GPIO is a 6-bit wide, bi-directional port. The corresponding data direction register is TRISIO. Setting a TRISIO bit (= 1) will make the corresponding GPIO pin an input (i.e., put the corresponding output driver in a Hi-impedance mode). Clearing a TRISIO bit (= 0) will make the corresponding GPIO pin an output (i.e., put the contents of the output latch on the selected pin). The exception is GP3, which is input only and its TRISIO bit will always read as ‘1’.

Reading the GPIO register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations.

Therefore, a write to a port implies that the port pins are read, this value is modified, and then written to the port data latch. GP3 reads '0' when MCLREN = 1. The TRISIO register controls the direction of the GP pins, even when they are being used as analog inputs. The user must ensure the bits in the TRISIO register are set when using them as analog inputs. I/O pins configured as analog inputs always read '0'.

Every GPIO pin has an interrupt-on-change option and every GPIO pin, except GP3, has a weak pull-up option. Control bits WPUx enable or disable each pull-up. Each weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset by the GPPU bit (OPTION<7>).

For enabled interrupt-on-change pins, the values are compared with the old value latched on the last read of GPIO. The 'mismatch' outputs of the last read are OR'd together to set the GP Port Change Interrupt flag bit (GPIF) in the INTCON register. This interrupt can wake the device from SLEEP. The user, in the Interrupt Service Routine, can clear the interrupt by reading or writing of GPIO, or by clearing the flag bit GPIF. A mismatch condition will continue to set flag bit GPIF. Reading GPIO will end the mismatch condition and allow the flag bit GPIF to be cleared.

Each GPIO pin is multiplexed with several other functions. The pins and their combined functions are briefly described below.

- GP0/AN0/CIN+
  - a general purpose I/O
  - an analog input for the A/D
  - an analog input to the comparator
- GP1/AN1/CIN-/VREF
  - as a general purpose I/O
  - an analog input for the A/D
  - an analog input to the comparator
  - a voltage reference input
- GP2/AN2/T0CKI/INT/COUT
  - a general purpose I/O
  - an analog input for the A/D
  - the clock input for TMR0
  - an external edge triggered interrupt
  - a digital output from the comparator
- GP3/MCLR/Vpp
  - a general purpose input
  - as Master Clear Reset

- GP4/AN3/T1G/OSC2/CLKOUT
  - a general purpose I/O
  - an analog input for the A/D (PIC12F675 only)
  - a TMR1 gate input
  - a crystal/resonator connection
  - a clock output

- GP5/T1CKI/OSC1/CLKIN
  - a general purpose I/O
  - a TMR1 clock input
  - a crystal/resonator connection
  - a clock input

## 1.6 PIC12F675 Timers

The timer/counter module Timer0 has the following features:

- 8-bit timer/counter
- Read/Write
- 8-bit software programmable prescaler
- Internal or external clock select
- Interrupt on overflow from FFh to 00h
- Edge select for external clock

Timer mode is selected by clearing the T0CS bit (OPTION_REG<5>). In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If TMR0 is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

Counter mode is selected by setting the T0CS bit (OPTION_REG<5>). In this mode, the Timer0 module will increment either on every rising or falling edge of Pin GP2/T0CKI. The incrementing edge is determined by the source edge (T0SE) control bit (OPTION_REG<4>). Clearing the T0SE bit selects the rising edge.

A Timer0 interrupt is generated when the TMR0 register timer/counter overflows from FFh to 00h. This overflow sets the T0IF bit. The interrupt can be masked by clearing the T0IE bit (INTCON<5>). The T0IF bit (INTCON<2>) must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt. The Timer0 interrupt does not wake the processor from SLEEP since the timer itself is shut-off during SLEEP.

An 8-bit counter is available as a prescaler for the Timer0 module, or as a post-scaler for the Watchdog Timer. The prescaler assignment is controlled in software by the control bit PSA (OPTION_REG<3>). Clearing the PSA bit will assign the prescaler to Timer0. Prescale values are selectable via the PS2:PS0 bits (OPTION_REG<2:0>). The prescaler is not accessible directly. When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g., CLRF 1, MOVWF 1, BSF 1, x....etc.) will clear the prescaler. When assigned to WDT, a CLRWDT instruction will clear the prescaler along with the Watchdog Timer. The prescaler assignment is under software control and it can be changed on the fly during program execution.

The PIC12F675 has a second timer module Timer1 with the following features:

- 16-bit timer/counter (TMR1H:TMR1L)
- Readable and writable
- Internal or external clock selection
- Synchronous or asynchronous operation
- Interrupt on overflow from FFFFh to 0000h
- Wake-up upon overflow (Asynchronous mode)
- Optional external enable input (T1G)
- Optional LP oscillator

The Timer1 Control register (T1CON) is used to enable/disable Timer1 and select the various features of the Timer1 module. There are three modes of operation for Timer1:

- 16-bit timer with prescaler
- 16-bit synchronous counter
- 16-bit asynchronous counter

In Timer mode, Timer1 is incremented on every instruction cycle. In Counter mode, Timer1 is incremented on the rising edge of the external clock input T1CKI. In addition, the Counter mode clock can be synchronized to the microcontroller system clock or run asynchronously. In Counter and Timer modules, the counter/timer clock can be gated by the T1G input. If an external clock oscillator is needed (and the microcontroller is using the INTOSC w/o CLKOUT), Timer1 can use the LP oscillator as a clock source.

The Timer1 register pair (TMR1H:TMR1L) increments to FFFFh and rolls over to 0000h. When Timer1 rolls over, the Timer1 interrupt flag bit (PIR1<0>) is set. To enable the interrupt on rollover, these bits must be set:

- Timer1 interrupt Enable bit (PIE1<0>)
- PEIE bit (INTCON<6>)
- GIE bit (INTCON<7>)

The interrupt is cleared by clearing the TMR1IF in the Interrupt Service Routine.

Timer1 has four prescaler options allowing 1, 2, 4, or 8 divisions of the clock input. The T1CKPS bits (T1CON<5:4>) control the prescale counter. The pre-scale counter is not directly read or written; and it is cleared upon a write to TMR1H or TMR1L.

Time1 is in asynchronous counter mode if control bit T1SYNC (T1CON<2>) is set. In this mode the timer increments without synchronizing to the internal clock. And it will continue to run during SLEEP mode and generate an interrupt on overflow.

A crystal oscillator circuit is built-in between pins OSC1 (input) and OSC2 (amplifier output). It is enabled by setting control bit T1OSCEN (T1CON<3>). The oscillator is a low power oscillator rated up to 37 kHz. It will continue to run during SLEEP. It is primarily intended for a 32 kHz crystal.

## 1.7 PIC12F675 Analog Comparator

The PIC12F675 has an analog comparator which compares the inputs at GP0 and GP1 by multiplexing. There is an on-chip Comparator Voltage Reference that can also be applied to an input of the comparator. The comparator output is read through the CMCON register which is read only. In addition, GP2 can be con-figured as the comparator output (digital). The Comparator Control Register (CMCON) contains the bits to control the comparator. The analog input must be between Vss and Vdd and the source impedance not larger than 10K ohms.

Both the comparator and voltage reference, if enabled before entering SLEEP mode, remain active during SLEEP. This results in higher SLEEP currents.

The comparator interrupt flag is set whenever there is a change in the output value of the comparator. Software will need to maintain information about the status of the output bits, as read from CMCON<6>, to determine the actual change that has occurred. The CMIF bit, PIR1<3>, is the comparator interrupt flag. This bit must be reset in software by clearing it to ‘0’. Since it is also possi-ble to write a '1' to this register, a simulated interrupt may be initiated.

## 1.8 PIC12F675 Analog-to-Digital Converter

The PIC12F675 has a 10-bit analog-to-digital converter (A/D). Four analog inputs are multiplexed into one sample and hold circuit. The output of the sample and hold is connected to the input of the converter. The converter generates a binary result via successive approximation and stores the result in a 10-bit register. The voltage reference used in the conversion is software selectable to either Vdd or a voltage applied by the Vref pin.

The ANS3:ANS0 bits (ANSEL<3:0>) and the TRISIO bits control the operation of the A/D port pins.

The A/D conversion cycle requires 11 Tad. The source of the conversion clock is software selectable via the ADCS bits (ANSEL<6:4>). There are seven possible clock options:

- Fosc/2
- Fosc/4
- Fosc/8
- Fosc/16
- Fosc/32
- Fosc/64
- FRC (dedicated internal RC oscillator)

For correct conversion, the A/D conversion clock (1/Tad) must be selected to ensure a minimum Tad of 1.6 µs. The minimum acquisition time is approximately 20 µs. The A/D converter module can operate during SLEEP.

## 1.9 PIC12F675 EEPROM Memory

The PIC12F675 has 128 bytes of EEPROM data memory. This memory is not directly mapped in the register file space. Instead, it is indirectly addressed through the Special Function Registers. There are four SFRs used to read and write this memory:

- EECON1
- EECON2 (not a physically implemented register)
- EEDATA
- EEADR

EEDATA holds the 8-bit data for read/write, and EEADR holds the address of the EEPROM location being accessed. A byte write automatically erases the location and writes the new data (erase before write). The write time is controlled by an on-chip timer. When the data memory is code protected, the CPU may continue to read and write the data EEPROM memory while the device

EECON1 is the control register with four low order bits physically implemented. The upper four bits are not implemented and read as '0's. Control bits RD and WR initiate read and write, respectively. These bits cannot be cleared, only set, in software. They are cleared in hardware at completion of the read or write operation. The WREN bit, when set, will allow a write operation. On power-up, the WREN bit is clear. EECON2 is not a physical register. Reading EECON2 will read all '0's. The EECON2 register is used exclusively in the Data EEPROM write sequence.

Data memory can be code protected by programming the CPD bit to ‘0’. When the data memory is code protected, the CPU is still able to read and write data to the Data EEPROM. It is recommended to code protect the program memory when code protecting data memory. This prevents anyone from programming zeroes over the existing code (which will execute as NOPs) to reach an added routine, programmed in unused program memory, which outputs the contents of data memory. Programming unused locations to ‘0’ will also help to prevent data memory code protection from being breached.

## 1.10 PIC12F675 Special Features

A number of features that make the PIC12F675 useful in real-time applications are also included:

- Oscillator selection
- RESET
  - Power-on Reset (POR)
  - Power-up Timer (PWRT)
  - Oscillator Start-up Timer (OST)
  - Brown-out Detect (BOD)
- Interrupts
- Watchdog Timer (WDT)
- SLEEP
- Code protection
- ID Locations
- In-Circuit Serial Programming

The PIC12F629/675 has a Watchdog Timer that is controlled by configuration bits. It runs off its own RC oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in RESET until the crystal oscillator is stable. The other is the Power-up Timer (PWRT), which provides a fixed delay of 72 ms (nominal) on power-up only, designed to keep the part in RESET while the power supply stabilizes. There is also circuitry to reset the device if a brown-out occurs, which can provide an at least 72 ms RESET. With these three functions on-chip, most applications need no external RESET circuitry. The SLEEP mode is designed to offer a very low current Power-down mode. The user can wake-up from SLEEP through:

- External RESET
- Watchdog Timer wake-up
- An interrupt

Several oscillator options are also made available to allow the part to fit the application. The INTOSC option saves system cost while the LP crystal option saves power.

If the code protection bit(s) have not been programmed, the on-chip program memory can be read out for verification purposes.

Four memory locations (2000h-2003h) are designated as ID locations where the user can store checksum or other code identification numbers. These locations are not accessible during normal execution but are read and written during Program/Verify phase. Only the Least Significant 7 bits of the ID locations are used.

In-Circuit Serial Programming is done with two lines for clock and data, and three other lines for:

- power
- ground
- programming voltage

This allows one to produce boards with unprogrammed devices, and then program the microcontroller just before use. It thus allows the most recent firmware or a custom firmware to be programmed. The device is placed into a Program/Verify mode by holding the GP0 and GP1 pins low, while raising the MCLR (Vpp) pin from VIL to VIHH. GP0 becomes the programming data and GP1 becomes the programming clock. Both GP0 and GP1 are Schmitt Trigger inputs in this mode. After RESET, to place the device into Programming/Verify mode, the program counter (PC) is at location 00h. A 6-bit command is then supplied to the device. Depending on the command, 14-bits of program data are then supplied to or from the device, depending on whether the command was a load or a read.

## 1.11 PIC12F675 Instruction Set

The PIC12F629/675 instruction set is highly orthogonal and is comprised of three basic categories:

- Byte-oriented operations
- Bit-oriented operations
- Literal and control operations

Each PIC12F629/675 instruction is a 14-bit word divided into an **opcode**, which specifies the instruction type, and one or more **operands**, which further specify the operation of the instruction.

One instruction cycle consists of four oscillator periods; for an oscillator frequency of 4 MHz, this gives a normal instruction execution time of 1 µs. All instructions are executed within a single instruction cycle, unless a conditional test is true, or the program counter is changed as a result of an instruction. When this occurs, the execution takes two instruction cycles, with the second cycle executed as a NOP.

Any instruction that specifies a file register as part of the instruction performs a Read-Modify-Write (R-M-W) operation. The register is read, the data is modified, and the result is stored according to either the instruction, or the destination designator ‘d’. A read operation is performed on a register even if the instruction writes to that register.

The following table is a summary of the instruction set:

| **Mnemonic** | | **Description** |
|---|---|---|
| BYTE-ORIENTED FILE REGISTER OPERATIONS | | |
| ADDWF | f, d | Add W and f |
| ANDWF | f, d | AND W with f |
| CLRF | f | Clear f |
| CLRW | - | Clear W |
| COMF | f, d | Complement f |
| DECF | f, d | Decrement f |
| DECFSZ | f, d | Decrement f, Skip if 0 |
| INCF | f, d | Increment f |
| INCFSZ | f, d | Increment f, Skip if 0 |
| IORWF | f, d | Inclusive OR W with f |
| MOVF | f, d | Move f |
| MOVWF | f | Move W to f |
| NOP | - | No Operation |
| RLF | f, d | Rotate Left f through Carry |
| RRF | f, d | Rotate Right f through Carry |
| SUBWF | f, d | Subtract W from f |
| SWAPF | f, d | Swap nibbles in f |
| XORWF | f, d | Exclusive OR W with f |
| BIT-ORIENTED FILE REGISTER OPERATIONS | | |
| BCF | f, b | Bit Clear f |
| BSF | f, b | Bit Set f |
| BTFSC | f, b | Bit Test f, Skip if Clear |

| | | |
|---|---|---|
| BTFSS | f, b | Bit Test f, Skip if Set |
| LITERAL AND CONTROL OPERATIONS | | |
| ADDLW | k | Add literal and W |
| ANDLW | k | AND literal with W |
| CALL | k | Call subroutine |
| CLRWDT | - | Clear Watchdog Timer |
| GOTO | k | Go to address |
| IORLW | k | Inclusive OR literal with W |
| MOVLW | k | Move literal to W |
| RETFIE | - | Return from interrupt |
| RETLW | k | Return with literal in W |
| RETURN | - | Return from Subroutine |
| SLEEP | - | Go into Standby mode |
| SUBLW | k | Subtract W from literal |
| XORLW | K | Exclusive OR literal with W |

The different fields are shown below:

| **Field** | **Description** |
|---|---|
| f | Register file address (0x00 to 0x7F) |
| W | Working register (accumulator) |
| b | Bit address within an 8-bit file register |
| k | Literal field, constant data or label |
| d | Destination select; d = 0: store result in W, d = 1: store result in file register f. Default 1 |

## 1.12 PIC12F675 Development Environment

Program development support from the manufacturer of PICmicro®, Microchip Technology Inc. is good. They consist of a number of hardware and software development tools:

- Integrated Development Environment
- Assemblers/Compilers/Linkers
- Simulators
- Emulators
- In-Circuit Debugger
- Device Programmers
- Low Cost Demonstration Boards
- Evaluation Kits

The manufacturer's web site (http://www.microchip.com) has very comprehensive information including product datasheets, product guides, application notes, prices and information on related products or systems.

Because of the popularity of these devices, there are now a host of third party suppliers for hardware and software products based on these MCUs. Amongst them is Basic Stamps produced by Parallax, Inc. (http://www.parallax.com).

The company released a stamp sized microcomputer using PIC chip that was programmed in BASIC. These BASIC Stamps have been very popular because of their ease of use, and relative low cost. It allows many beginners to program a microcontroller for the first time. Parallax's customers include everyone from scientists and hobbyists to engineers and entrepreneurs. By 2002 there were over three million BASIC Stamps in use.

There are also books aiming at introducing these devices and their use. John Iovine's "PIC Microcontroller Project Book" (McGraw-Hill, 2000, ISBN 0-07-135479-4) is an introductory book on the subject. It does not assume any prior knowledge on microcontrollers. The author simplifies the programming of these devices by using a BASIC compiler produced by microEngineering Lab, Inc. (http://www.melabs.com). A dozen simple projects are described in the book, including simple I/O control, speech synthesizer, LCD display, sensor monitoring and motor control.

While the majority of compilers and development environment for PIC are MS Windows or DOS based, there are now versions available under Linux environment. Information can be found at:
http://www.eg3.com/WebID/embedded/pic/blank/perspage/1-a-p.htm

Finally, the cost of the PIC12F675 varies from about $1.25 to $1.65, depending on the type of packaging. A PDIP 300mil package type (PIC12F675-I/P) is available at $1.25 per piece.

## 1.13 PIC Application – Motion Detector

Small microcontrollers such as the PICs are well suited for a large range of applications in laboratories. They are particular useful in implementing small real-time applications. Real-time requirements mean tasks must be executed at precise times or over specific durations. Such requirements are difficult to fulfil if there are a large number of concurrent tasks in the system. However, if one can partition the system into small modules or sub-systems, the problem is much simple to handle. This is similar to the principle of encapsulation used in object oriented programming.

Many tasks that requires very stringent timing, e.g. monitoring the output of a sensor accurately every second and doing so for exactly 10 ms each time, are very easy to implement using small microcontrollers which have crystal oscillators, timers and perhaps real-time clock.

A one dollar microcontroller can easily resolve time to within microseconds and typically has long term time stability of one part in $10^5$ using crystal oscillator. In the event that this produces unacceptable drift in a distributed environment, one simple solution is to synchronous with a master clock in the system, or implement isochronous clock for the entire distributed system.

This section describes a small project using a PIC12LC672 microcontroller to illustrative the simplicity in using this type of device. The timing requirement in the project makes it a small real-time system in a way. The objective is to construct a simple, lightweight and low cost motion detector. There are many situations when movements of personnel, animals or other objects in rooms under normal lighting conditions may be of interest. And there are many ways of detecting such motions, ranging from applying video pattern recognition to using mechanical sensors or accelerometers.

A conceptually very simple method is to monitor the change of light intensity with a sensor affixed to the object of interest, and hence reduce the status of whether the object is stationary or moving. This method is attractive because cadmium sulphide (CdS) light dependent resistors (LDRs) can be used as light sensor and they are small, low cost, readily available and easy to handle. Several LDRs are shown in the picture below; the smallest one has a diameter of about 4 mm and costs less than $1 each. Unfortunately, a practical motion detector using a LDR requires a fair bit of signal processing. Before the arrival of low cost microcontrollers such as PICs, analogue and digital circuits were used for this signal processing. The situation has changed dramatically with the use of small microcontroller as embedded processor. Most of the signal conditioning can be done in software now.

The common LDR has a nonlinear resistance range of just under 1kΩ (at 10 lux) to 20 MΩ (dark). These are five orders of magnitude change, which is good. However, in normal use, one cannot expect changes from 10 lux to complete darkness when moving about in a room! The resultant resistance change while one is moving about is considerably smaller. By having a tunnel or telescopic view from the LDR, the light and hence resistance change can be enhanced to a certain extent. However, there is a limit to the improvement possible. An overall change of 0.1 to 10 times in resistance is more likely the case. This magnitude of change is sufficiently large for an 8-bit ADC to be used, since a resolution of 256 levels can be achieved with 8 bits.

The difficulty lies in the dynamic range of the ambient light intensity. Even in offices, the brightness of areas under lamps, when sunlight shines through windows and at dark corners varies considerably, resulting in orders of magnitude change in resistance. Thus when an 8-bit ADC is used in a straightforward manner, it will not function properly. If a large dynamic range is set, it is insensitive to changes due to normal movement. On the other hand, if the dynamic range is small, it will not work in bright or dark areas which produce extreme resistance values.

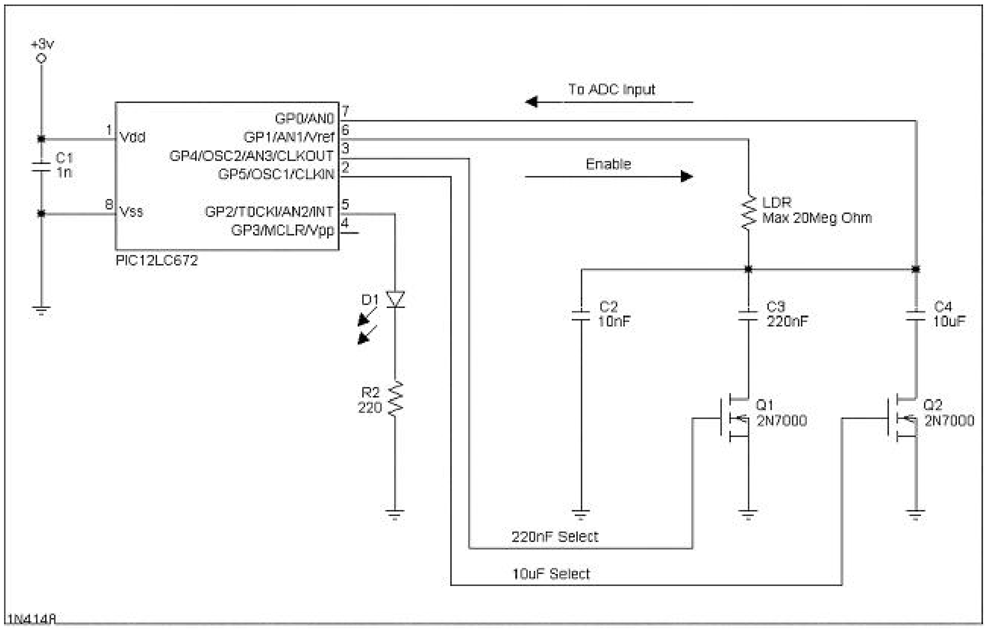
One possible solution to the dynamic range problem is to use potential divider arrangement with a fixed resistor connected in series with the LDR. The potential across one of the resistors is measured with the ADC of the microcontroller. By switching in different fixed resistors, one can cover practically any range of resistance value of the LDR. There are however a couple of shortcomings with this arrangement. First, the fixed resistor will typically have resistance value of the same order of magnitude as the LDR. This may mean unnecessary drain on power, which has to be kept as small as possible for such battery operated devices. Second, most modern lighting in offices uses fluorescent lamps which produce fluctuating light intensity at twice the mains frequency (f) due to the mercury discharge characteristic. Thus when the mains is 50-Hz, the light in a room actually various between zero and maximum every 10 ms or half the period (t= 1/f), even though not discernable to us due to persistence of vision of our eyes. This, however, is detected by the LDR. Spot measurement of the resistance of LDR thus reflects both flickering of the fluorescent lamp (when measurement is taken) and motion of the sensor (where the sensor points to).

An average or integration over multiple t (half period) is needed to overcome this problem. This can be done by accurate sampling many times within the period mentioned and take the average.

There is however a better solution to the problem. An integration can be done with ease by using an RC charging technique. Instead of using a DC potential arrangement, a capacitor is connected to the LDR in series. By charging the capacitor via the LDR over a period of t or multiple t, the voltage on the capacitor is an indication of the integration over the period. This will eliminate the fluctuating light output problem of fluorescent lamps.

If different capacitances can be switched in, the dynamic range of five orders of magnitude can be achieved quick easily. This arrangement has the added advantage that even when the LDR resistance is low, the current through the circuit is kept low, limited by the charging characteristic of the RC circuit.

A practical arrangement using the above technique for motion detection is shown below. The circuit consists of a PIC12LC672, an LDR, an LED, two MOSFETs, 4 capacitors and a resistor. The whole circuit is powered by a 3 V lithium ion coin battery such as the CR2032. One output pin GP2 is used to control the indicator LED; the series resistor R2 limits the current through the LED to several mA. The Vref output provides the charging current through the LDR while the voltage across the capacitor(s) is measured by analog input pin AN1. GP0 is configured as an input to discharge the capacitors. GP4 and GP5 act as output, controlling the MOSFET switches for capacitors C3 and C4.



It can be seen that the LDR is always connected in series with C2 to ground. Two other capacitors C3 and C4 may be switched in (in parallel to C2) via MOS-FET Q1 and Q2 respectively. The voltage across the capacitors and ground is measured by the ADC of the microcontroller. Vref controls the charging of the capacitor(s). Normally it is kept at 0 V. When a measurement is needed, it is turned to high for t second thus charging the capacitor(s) for exactly half the period of the mains. The ADC reading is taken and the resistance of the LDR can be deduced.

Adaptive measurement to cover a wide dynamic range by selectively switching in C3 or C4 when necessary. When it is bright and LDR resistance low, C2 will charged up to saturation rapidly, well within t second. If this condition is detected, C3 is switched in before the next sampling. The same procedure is repeated for C4 if necessary. Similarly, C3 and C4 may be switch out when the ADC reading is too low, indicating too high a LDR resistance.

After each measurement, the capacitors are discharged to 0V by turning GP0 to low.

The LED acts as an indicator in this simple example. It is turned on momentarily when motion is detected.

Since light intensity is sensed in this circuit, we can use it to monitor changing lighting condition. Interesting enough, this may pose a problem when the circuit is used as a motion detector. Changing ambient light intensity due of switching on or off of lamps, gradual brightening or dimming of rooms due to external light variation (e.g. sun rising) may be misconstrued as resulting from motion. This can be avoided by appropriate algorithms. Gradual changing of room brightness may be software filtered. The rate of change of the light input can be used as a criterion. The sudden switching on and off of lamps will produce strictly step changes in light intensity which again can be identified by software.

# 2 Microprocessors by Rabbit Semiconductor

A Z-80 based high performance 8-bit microcontroller Rabbit 2000® was first introduced in 1999 by Rabbit Semiconductor, a fabless semiconductor company in Davis, California. The subsequent RabbitCore line of products introduced in 2001 has been very well received and it is one of the most popular MCU modules for embedded systems. Now a relatively wide range of cores are available that are very cost effective while offering a wide variety of form factors, memory capacities and functionalities.

Amongst the major attractions of this family of processors are the relatively large memory size of 1 Mbytes, Ethernet and TCP/IP support, and an integrated software development environment using Dynamic C.

Like other semiconductor manufacturers, Rabbit Semiconductor offers a range of products from simple components to full development systems. They are Rabbit Microprocessors, Microprocessor Core Modules, Development Kits, Peripherals and Accessories, and Software. Due to the complexity of the range of Rabbit Microprocessors, they are no longer simple to implement. These processors, by virtue of their processing power and resources, are generally large chips with 100 or 128 leads using quad flat package (QFP) or ball grid array (BGA) housings. These surface mount packages are generally beyond the means of ordinary laboratories that are not working in microelectronics.

Microprocessor core modules that contain the processor and necessary peripheral components including connectors on a small PCB are therefore more suitable for laboratory use. It turns out that the core modules produced by manufacturers such are Rabbit Semiconductor are generally very cost effective. We shall therefore introduce the core modules instead of the basic processors.

## 2.1 RabbitCores

Two series of core modules based on the Rabbit 2000 and 3000 processors respectively are produced to cater for a wide range of applications and requirements. The RCM2000 cores have the following main features:

- Rabbit 2000 processor
- 256K Flash memory
- 128-512K SRAM
- 40 general purpose I/O
- 18 – 25 MHz clock
- 1.9” x 2.3” x 0.5”

- 5V operation

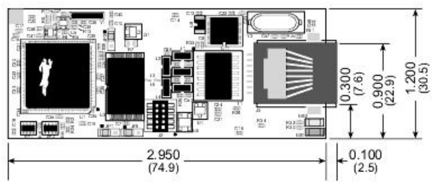
The smallest member, RCM2020 with 128K SRAM, operates at 18.432 MHz and is retailing at $39 per unit while the RCM2000 with 256K SRAM and 25.8MHz clock is at $69. These are extremely powerful modules requiring very few external components for many laboratory or other applications.

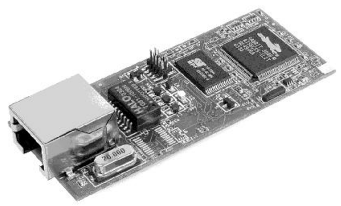
One of the major attractions of the RabbitCores is the incorporation of 10Base-T Ethernet capability including an RJ45 socket, as in the case of RCM2200 series. Based on the same processor, the other features are similar, other than a smaller number (26) of I/O pins are available to the users.

The high end series by Rabbit Semiconductor is the RCM3000. This series has 10Base-T Ethernet, up to 512K each of Flash and SRAM, encoder inputs, PWM outputs, and pulse capture and measurement capabilities. Two 34-pin connection headers provide 52 digital I/O shared with 6 serial ports and alternate I/O features. It also features a battery-backed real-time clock, glueless memory and I/O interfacing, and low power "sleepy" modes (<2mA). A fully enabled slave port permits easy master-slave interfacing with another processor-based system, and an alternate I/O bus can be configured for 8 data lines and 6 address lines sharing with parallel I/O.

## 2.2 RCM3700 Hardware

We shall now look at the hardware of one of the RabbitCores, RCM3700, in more detail. A picture and the dimensions are shown below. The main hardware and physical specifications are given in the manufacturer's datasheet and are tabulated below for reference.



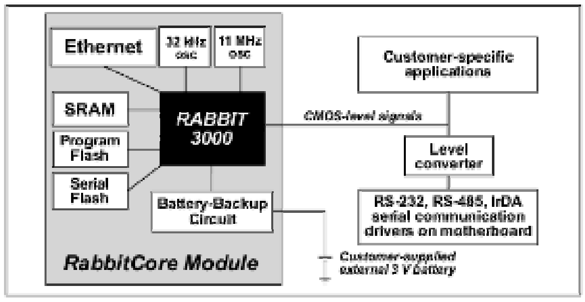


| Parameter | RCM3700 | RCM3710 |
|---|---|---|
| Microprocessor | Low-EMI Rabbit 3000 at 22.1 MHz | |
| Ethernet Port | 10Base-T, RJ-45, 2 LEDs | |

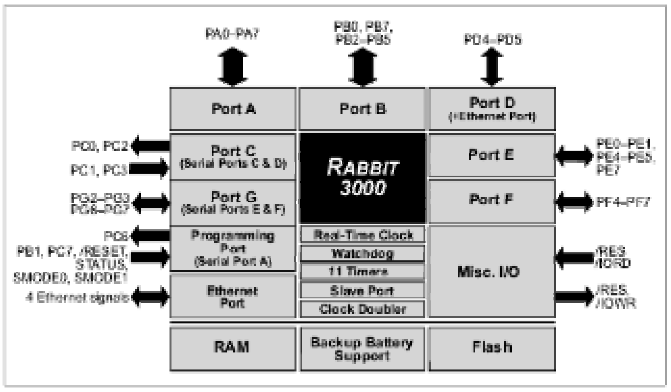
| | | |
|---|---|---|
| Flash Memory | 512K | 256K |
| SRAM | 512K | 256K |
| Serial Flash Memory | 1Mbyte | |
| Backup Battery | Connection for user-supplied backup battery (to support RTC and SRAM) | |
| General-Purpose I/O | 33 parallel digital I/0 lines:<br>• 31 configurable I/O<br>• 2 fixed outputs | |
| Additional I/O | Reset | |
| Auxiliary I/O Bus | Can be configured for 8 data lines and 5 address lines (shared with parallel I/O lines), plus I/O read/write | |
| Serial Ports | Four 3.3 V CMOS-compatible ports configurable as:<br>• 4 asynchronous serial ports (with IrDA) or<br>• 3 clocked serial ports (SPI) plus 1 HDLC (with IrDA) or<br>• 1 clocked serial port (SPI) plus 2 HDLC serial ports (with IrDA) | |
| Serial Rate | Maximum asynchronous baud rate = CLK/8 | |
| Slave Interface | A slave port allows the RCM3700 to be used as an intelligent peripheral device slaved to a master processor, which may either be another Rabbit 3000 or any other type of processor | |
| Real-Time Clock | Yes | |
| Timers | Ten 8-bit timers (6 cascadable), one 10-bit timer with 2 match registers | |
| Watchdog/Supervisor | Yes | |
| Pulse-Width Modulators | 4 PWM output channels with 10-bit free-running counter and priority interrupts | |

| | | |
|---|---|---|
| Input Capture/ Quadrature Decoder | 2-channel input capture can be used to time input signals from various port pins<br>• 1 quadrature decoder unit accepts inputs from external incremental encoder modules or<br>• 1 quadrature decoder unit shared with 2 PWM channels | |
| Power | 4.75-5.25 V DC<br>100 mA @ 22.1 MHz, 5 V; 78 mA @ 11.05 MHz, 5 V | |
| Operating Temperature | -40°C to +70°C | |
| Humidity | 5% to 95%, noncondensing | |
| Connectors | One 2 x 20, 0.1" pitch | |
| Board Size | 1.20" x 2.95"" x 0.88"<br>(30 mm x 75 mm x 22 mm) | |
| Price | $59 | $49 |

A simplified block diagram of the RCM3700 is show below:



The various ports and I/O lines are shown in the following diagram.



Most of the I/O lines on the Rabbit 3000 microprocessor are dual function. The factory defaults and the alternate configurations are shown in the following table:

| Pin | | Pin Name | Default Use | Alternate Use | Notes |
|---|---|---|---|---|---|
| Header J1 | 1-8 | PA[7:0] | Parallel I/O | External data bus (ID0-ID7) Slave port data bus (SD0-SD7) | External Data Bus |
| | 9 | PF1 | Input/Output | QD1A CLKC | |
| | 10 | PF0 | Input/Output | QD1B CLKD | |
| | 11 | PB0 | Input/Output | CLKB | |
| | 12 | PB2 | Input/Output | IA0 /SWR | External Address 0 Slave port write |
| | 13 | PB3 | Input/Output | IA1 /SRD | External Address 1 Slave port read |

| | | | | | |
|---|---|---|---|---|---|
| | 14 | PB4 | Input/Output | IA2<br>SA0 | External Address 2<br>Slave Port Address 0 |
| | 15 | PB5 | Input/Output | IA3<br>SA1 | External Address 3<br>Slave Port Address 1 |
| | 16 | PB7 | Input/Output | IA5<br>/SLAVEATTN | External Address 5<br>Slave Port Attention |
| | 17 | PF4 | Input/Output | AQD1B<br>PWM0 | |
| | 18 | PF5 | Input/Output | AQD1A<br>PWM1 | |
| | 19 | PF6 | Input/Output | AQD2B<br>PWM2 | |
| | 20 | PF7 | Input/Output | AQD2A<br>PWM3 | |
| | 21 | PC0 | Output | TXD | Serial Port D |
| | 22 | PC1/PG2 | Input/Output | RXD/TXF | Serial Port D<br>Serial Port F |
| | 23 | PC2 | Output | TXC | Serial Port C |
| | 24 | PC3/PG3 | Input/Output | RXC/RXF | Serial Port C<br>Serial Port F |

| | | | | | |
|---|---|---|---|---|---|
| | 25 | PE7 | Input/Output | I7<br>/SCS | External Address 7 Slave Port Chip Select |
| Header J1 | 26 | PE5 | Input/Output | I5<br>INT1B | |
| | 27 | PE4 | Input/Output | I4<br>INT0B | |
| | 28 | PE1 | Input/Output | I1<br>INT1A | I/O Strobe 1<br>Interrupt 1A |
| | 29 | PE0 | Input/Output | I0<br>INT0A | I/O Strobe 0<br>Interrupt 0A |
| | 30 | PG7 | Input/Output | RXE | Serial Port E |
| | 31 | PG6 | Input/Output | TXE | |
| | 32 | /IOWR | Output | | External write strobe |
| | 33 | /IORD | Input | | External read strobe |
| | 34 | PD4 | Input/Output | ATXB | Alternate Serial Port B |
| | 35 | PD5 | Input/Output | ARXB | |
| | 36 | /RES | Reset output | Reset input | Reset output from Reset Generator |
| | 37 | VBAT | | | |
| | 38 | GND | | | |
| | 39 | +5 V | | | |
| | 40 | GND | | | |

## 2.3 RCM3700 Memory & I/O Bus

The core module typically comes with onboard memories large enough for your application. In this case they are 512 KB of flash memory and 512 KB of SRAM (RCM3700) or 256 KB each (RCM3710). A Flash Memory Bank Select jumper configuration option based on 0 Ω surface-mounted resistors exists at header JP1 on the RCM3700 modules. This option, used in conjunction with some configuration macros, allows Dynamic C to compile two different co-resident programs for the upper and lower halves of the 512K flash in such a way that both programs start at logical address 0000. This is useful for applications that require a resident download manager and a separate downloaded program.

An additional 1 MB of serial flash memory is available for mass storage purposes such as data and web pages.

Thus the address lines (A0-A18) and all the data lines (D0-D7) are routed internally to the onboard flash memory and SRAM chips. However, I/0 write (/IOWR) and I/0 read (/IORD) are available for interfacing to external devices.

Parallel Port A can be used as an external I/O data bus to isolate external I/O from the main data bus. Parallel Port B pins PB2-PB5 and PB7 can also be used as an auxiliary address bus.

It is noted that /RES is an output from the reset circuitry that can be used to reset other peripheral devices. This pin can also be used to reset the microprocessor.

## 2.4 RCM3700 Serial, Ethernet & Programming Ports

The core module provides five serial ports designated as Serial Ports A, C, D, E, and F. All five serial ports can operate in an asynchronous mode up to the baud rate of the system clock divided by 8. An asynchronous port can handle 7 or 8 data bits. A 9th bit address scheme, where an additional bit is sent to mark the first byte of a message, is also supported.

Serial Port A is normally used as a programming port, but may be used either as an asynchronous or as a clocked serial port once the RCM3700 has been programmed and is operating in the Run Mode.

Serial Ports C and D can also be operated in the clocked serial mode. In this mode, a clock line synchronously clocks the data in or out. Either of the two communicating devices can supply the clock. Serial Ports E and F can also be configured as HDLC serial ports. The IrDA protocol is also supported in SDLC format by these two ports.

It is noted that Serial Ports C and D, and Serial Port F share some common pins on header J1 and hence only one of the two can be used at any one time. The choice of port(s) depends on the application. Serial Ports C and D are for clock serial links while Serial Port F is for an HDLC serial link.

The core module has an Ethernet port (J3) complete with an RJ45 socket. Two LEDs are placed next to the RJ-45 Ethernet socket, one to indicate an Ethernet link (LINK) and one to indicate Ethernet activity (ACT). The RJ-45 connector is shielded to minimize EMI effects to/from the Ethernet signals.

Serial Port A has special features that allow it to cold-boot the system after reset. Serial Port A is also the port that is used for software development under Dynamic C. The RCM3700 is accessed using a 10-pin program header labeled J2. The programming port uses the Rabbit 3000's Serial Port A for communication, and is used for the following operations:

- Programming/debugging
- Cloning
- Remote program download/debug over an Ethernet connection

The Rabbit 3000 startup-mode pins (SMODE0, SMODE1) are available at the programming port so that an externally connected device can force the RCM3700 to start up in an external bootstrap mode. The RCM3700 can be reset by Dynamic C via the /RESET line on the programming port.

The clock line for Serial Port A is presented to the programming port, which makes synchronous serial communication possible.

The programming port is used to start the RCM3700 in a mode where the RCM3700 will download a program from the port and then execute the program. It is also used during debugging. The programming port may also be used as an application port when necessary.

All three clocked Serial Port A signals are available as

- a synchronous serial port
- an asynchronous serial port, with the clock line usable as a general CMOS input
- two general-purpose CMOS inputs and one general-purpose CMOS output.

The two startup-mode pins, SMODE0 and SMODE1, are available as general CMOS inputs after they are read during the initial boot-up. The logic state of these two pins determines the startup procedure after a reset. The status pin may also be used as a general-purpose CMOS output.

For convenience, the RCM3700 is automatically set into program mode when the PROG connector on the programming cable is attached, and automatically into run mode when no programming cable is attached. The DIAG connector of the programming cable may be used on header J2 of the RCM3700 Prototyping Board with the RCM3700 operating in the run mode. This allows the programming port to be used as a regular serial port.

It is noted that the RCM3700 board does not have any serial transceivers directly on the board. These transceivers may be incorporated on the motherboard the RCM3700 is mounted on. The manufacturer has Prototyping Boards with RS232, RS-485 and IrDA transceiver chips mounted. However, it is quite straightforward to build your own.

## 2.5 RCM3700 Special Features

The RCM3700 uses the Rabbit 3000 microprocessor's internal clock doubler that allows half-frequency crystals to be used to reduce radiated emissions. The 22.1 MHz frequency specified for the RCM3700 is generated using an 11.06 MHz resonator. The clock doubler may be disabled if 22.1 MHz clock speed is not required. This will reduce power consumption and reduce radiated emissions.

Another special feature is spectrum spreading. The Rabbit 3000 uses a spectrum spreader unit that modifies the clock by shortening and lengthening clock cycles. The effect of this is to spread the spectral energy of the clock harmonics over a fairly wide range of frequencies. This limits the peak energy of the harmonics and reduces EMI that may interfere with other devices as well as reducing the readings in government mandated EMI tests. The spectrum spreader has two operating modes, normal spreading and strong spreading. The spreader can also be turned off.

## 2.6 A RabbitCore Application – Mass Storage using MultiMediaCard (MMC)

In a later section of this series of lectures, a demonstration system is set up using a Rabbit Core as a web server accessible in the Internet. The Rabbit Core acts as an embedded Ethernet Module talking to two separate legacy boards (HC11) via an RS485 network. Each legacy board is a thermometer using a thermistor with a 555 timer circuit to measure resistance. It demonstrates quite a few elements in a distributed laboratory instrumentation system. In this section an memory extension to the RabbitCore is described. It is hoped that it provides some useful information for those who require mass data storage in small embedded systems.

Embedded processors such as RabbitCores and TINIs are really powerful work-horses in embedded systems around laboratories. Many laboratory experiments and projects do not require any more processing power than what is available in these processors. However, being 8-bit processors, these systems usually lacks mass storage capabilities. RabbitCore is typically limited to 1MB of Flash and SRAM. The inclusion of an additional 1 MB of serial Flash memory in the newer core such as RCM3700 is a much welcomed feature.

There are many situations when more mass storage capacity is required. A field data logging system is a classic example. While the process of data logging is straightforward and can be handled by a RabbitCore easily, the amount of data to be collected often prove too much for such a core module. If the embedded system can be networked to other computers, data collected by the former can be transferred to the mass storage of a PC. A better solution is to implement local mass storage on the embedded system itself.

In PCs, hard disks have been standard mass storage all along. Unfortunately, connecting standard hard disks to a RabbitCore system, while tempting in terms of costs, is rather complicated in both software and hardware. One is thus forced to use the PC itself as a secondary storage. This is acceptable in many applications. However, it leaves an umbilical cord between the embedded system and the PC.

The recent crop of miniature USB Flash memories, emulating disk drives (called by different names including Pen Drive) in PC systems, is another tempting solution. Unfortunately, yet again, the implementation of USB host in systems like RabbitCore or TINI is difficult if not practically impossible.

There is another solution though. It turns out that commercially available Flash memories designed typically for consumer products such as digital cameras are readily available and are affordable. For a few tens of dollars one can get a 128 Mbytes of Flash card. There are several types of such cards in the market. One candidate best suited for interfacing to a small embedded system is the MultiMediaCard (MMC) because it requires only a few lines for interfacing. A high speed serial interface using Serial Peripheral Interface (SPI) allows data to be transferred.

This section describes a small project to incorporate an MMC mass storage device on a RabbitCore thereby enhancing the power of this 8-bit embedded processor significantly. With the possible of having 512 MB of removable storage on the RabbitCore the scope of application instantly widens enormously.

The MMC, introduced by SanDisk and Siemens/Infineon in 1997, is a highly integrated flash memory that is accessed via a dedicated serial interface, optimized for fast (up to 20Mbps) and reliable data transmission. It is a low-power device with a very small form factor (32mm x 24mm x 1.4mm) weighing approximately 1.5g. It has been chosen as the storage medium of choice because it provides an inexpensive, mechanically robust storage medium in card form, yet having a simple interface with minimal hardware requirements.

The MMC includes an on-card microcontroller, which manages the interface protocols, data storage and retrieval, error correction code, defect handling, diagnostics, power management and clock control. All device and interface configuration data are stored on the MMC itself. It has a seven pin serial interface, which allows for easy integration into any design regardless of the microprocessor/microcontroller used.

The standard MMC interface with external processor implements the MultiMediaCard protocol, which offers a high performance serial link designed for maximum scalability and configurability. In addition to the MMC interface, the MMC also supports an alternate communication protocol based on the Serial Peripheral Interface (SPI) standard. The SPI standard defines the physical link only, and not the complete data transfer protocol. The MMC SPI implementation uses a subset of the MultiMediaCard protocol and command set. It is intended to be used by systems that require a small number of cards (typically one) and have lower data transfer rates compared to MultiMediaCard protocol based systems. A comparison of these two modes of communication and their different pin assignments are given in the two tables below.

**MultiMediaCard/SPI Comparison**

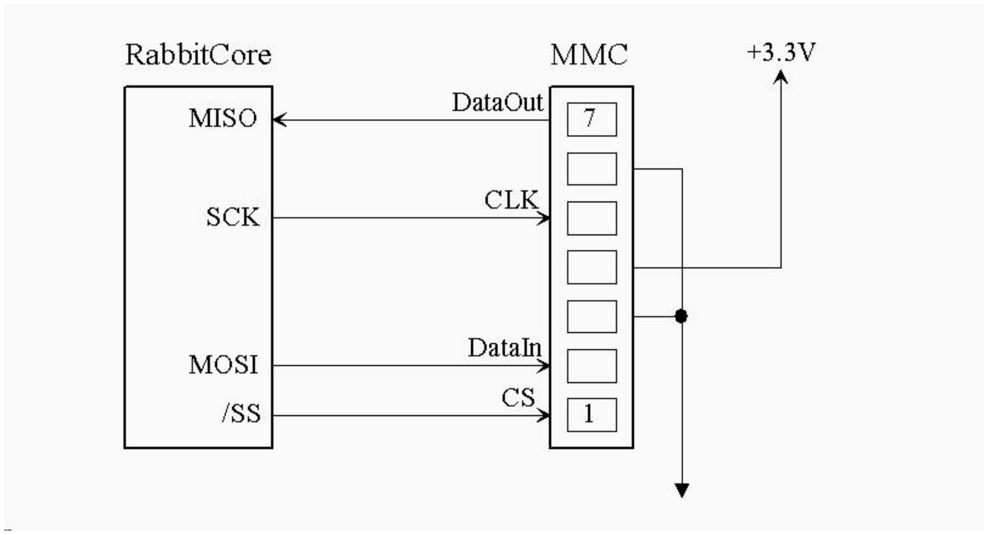
| *MultiMediaCard* | *SPI* |
|---|---|
| Three-wire serial data bus (Clock, Command, Data) | Three-wire serial data bus (Clock, Data In, Data Out) + card specific chip-select signal |
| Up to 64k cards addressable by the bus protocol | Card selection via a hardware chip- |

| *MultiMediaCard* | *SPI* |
|---|---|
| | select signal |
| Easy card identification | Not available |
| Error protected data transfer | Optional |
| Sequential and single/multiple block oriented data transfer | Single/multiple block read/write |

**MultiMediaCard Pad Definition**

| *Pin #* | *MultiMediaCard Mode* | | *SPI Mode* | |
|---|---|---|---|---|
| | **Name** | **Description** | **Name** | **Description** |
| 1 | RSV | Reserved for future use | CS | Chip Select |
| 2 | CMD | Command/Response | DI | Data In |
| 3 | VSS1 | Ground | VSS1 | Ground |
| 4 | VDD | Supply voltage | VDD | Supply voltage |
| 5 | CLK | Clock | SCLK | Clock |
| 6 | VSS2 | Ground | VSS2 | Ground |
| 7 | DAT | Data | DO | Data Out |

Although the SPI mode has a lesser set of features, the design effort is less. We use this mode in the RabbitCore implementation. The interface between the MMC and the RabbitCore is shown below.



As can be seen above, the hardware interface between the MMC and the host processor is rather straightforward. The program required to implement the basic communication with the MMC in SPI mode is relatively simple. In SPI mode, the communication always starts by driving the /SS line low and keeping it in that state until a complete MMC-host processor two-way transaction is done. All commands to the card are six bytes long. The first byte is always the “command” byte. After the command byte is sent, it must always be followed by four argument or address bytes, with the most significant byte sent first. Even if the command has no arguments, four “null” or “don't care” bytes must be sent. The last byte to be sent is the CRC byte. In SPI mode, the default mode of operation is “non-protected”. This means that CRC calculation is disabled and the command protocol can be satisfied by sending an $FF in the CRC byte. As in the case of the argument bytes, the CRC byte must be sent even if CRC checking is disabled to ensure that the command sequence is six bytes long.

| | | | | | |
|---|---|---|---|---|---|
| CMD | ARG1 (MSB) | ARG2 | ARG3 | ARG4 (LSB) | CRC |

The challenge in incorporating the MMC in a RabbitCore system is not in the device level program, but rather in the implementation of a file system to be used with it. Custom file systems may be implemented for simple storage requirements in a closed system. However, in many applications, it is often desirable to be able to directly access the data written to the MMC from a PC. Thus a standard file system should be implemented if possible.

# 3 ETRAX System-on-Chip by Axis Semiconductor

While RabbitCores and TINIs are very powerful and low cost microcomputers that are very useful in a lot of embedded system application, there are nevertheless limitations in terms of memory and processing power when compared to standard desktop PCs. Thus the last few years saw several manufacturers announced or attempted to bring up truly small and highly integrated modules that will run operating systems such as Linux and have memory and peripheral devices similar to those of PCs. System on a chip in other words. One such chips is available now – the ETRAX 100LX MCM 2+8 by Axis Semiconductor.

The history of the this chip dates back to 1990s when Axis Semiconductor designed a series of ASIC chips for their business in the IBM mainframe and printer protocol converter market. ETRAX which is an acronym of Ethernet, Token Ring, Axis, is a 32-bit RISC chip first appeared in 1993, the first system-on-a-chip for networking applications. Subsequent development led to the inclusion of SCSI and memories.

The companies now market two chips. The ETRAX 100LX has a 100 Mbps Ethernet and runs at 100 MIPS, besides having MMU, USB, synchronous serial ports and SDRAM support. The most sophisticated member is the ETRAX 100LX MCM 2+8. In this chip there are 2 MB flash and 8 MB SDRAM which will run embedded Linux. With a built in MMU, ETRAX 100LX runs Linux 2.4 kernel as is, without resorting to uClinux patches. More memories are being planned and shall be available soon.

## 3.1 ETRAX 100LX

This 100-MIPS 32-bit 0.25µm RISC chip has very impressive specifications:

- 32 bit RISC CPU core
- 10/100 Mbps Ethernet controller
- 4 asynchronous serial ports
- 2 synchronous serial ports
- 2 USB ports
- 2 Parallel ports
- 4 ATA (IDE) ports
- 2 Narrow SCSI ports (or 1 Wide)
- Support for SDRAM, Flash, EEPROM, SRAM
- $40

The CPU has 32-bit data and address format and has 15 general purpose 32-bit registers. The instruction set is optimised for 16/32 bit operation. The clock speed is 100 MHz. There are User and Supervisor mode for restricted access and selection of appropriate address mapping by the MMU.

The Memory Management Unit (MMU) is really a feature that distinguishes this type of CPU from smaller ones. The MMU allows 4 GB of virtual address space for each user process with space protection. There is also an 8-KB cache memory.

Network access capability is of course another key feature in the new crop of embedded chips. In ETRAX, the Ethernet controller supports both 100 Mbps and 10 Mbps, compatible with IEEE 802.3 and Fast Ethernet standards.

A high speed CPU is not sufficient if the system has to do any data handling jobs. Direct memory access is a must. There are 10 DMA channels each with 64 bytes FIFO for low latency and high throughput data transfer from internal and external units. A total of 200 MBytes per second bandwidth is available. DMA and cache feature helps to allow bursty access to and from memory to take full advantage of SDRAM and EDO DRAM performance.

The four built in asynchronous serial ports operate over a wide range of speed. The internal baud rate is programmable from 48 Hz to 1.5625 MHz and the external baud rate can be double that. Fixed baud rates are available from 300 Hz to 1.8432 MHz and at 6.25 MHz which is non-standard. Synchronous serial transfer is possible with clock rates from 32 kHz to 4.096 MHz. There are two such ports.

As serial connection is rapidly dominated by Universal Serial Bus (USB), the EXTRAX offers two such ports. It supports USB 1.1 Host and Device mode operation for control and bulk traffic only. The hardware supports dynamic connect/disconnect, suspend/resume and remote wakeup.

Two parallel I/O ports are available and can be used through register access or internal DMA. Many protocols are supported:

- IBM XT/AT compatible Centronics
- IBM PS/2 compatible Centronics
- HP Fast Mode
- IBM Fastbyte
- IEEE 1284 – byte, nibble, ECP and EPP
- Parallel port wide – 16 bit, 12 MBytes per second

ETRAX supports either two 8-bit or one 16-bit SCSI interface in both synchronous and asynchronous modes, including SCSI-3 and FAST-20. However, external bus drivers are needed. This bus is multiplexed on the same pins as the EIDE/ATA-2 ports, the parallel ports and two of the serial ports.

To support disk drives and other devices using the EIDE/ATA-2 interface, the ETRAX supports 4 such ports or up to 8 disk drives. This bus is on the same pins as the SCSI, the parallel ports and two of the serial ports.

The built-in bus interface and memory controllers work with SDRAM, EDO DRAM, SRAM, EPROM, parallel EEPROM and FlahsPROM. Thu bus can be configured to 16 or 32 bits. It also supports 64-bit SDRAM DIMM and SO-DIMM modules, Double Data Rate SDRAM, and power save mode.

Bootstrapping is a necessary feature. The ETRAX supports loading to internal cache memory from parallel port, serial port, and network interface. Code, loaded to cache, can subsequently be used to enable the downloading of program to Flash PROM or other external memory.

In most embedded system applications, timers and watchdog are essential for numerous real-time operation and to ensure that the system can reset itself in exceptional circumstances. There are two 8-bit timer with programmable clock from 381 Hz to 12.5 MHz. Or fixed timer clocks from 300 Hz to 1.842 MHz, and a nonstandard clock at 6.25 MHz. There is also a watchdog timer.

Other features of the EXTRAX includes two general purpose port each having 8 programmable I/O pins. The internal clock of 100 MHz is derived from a 30-MHz external clock using PLL technique. Interrupt controls include internal interrupts due to I/O ports, network interface, DMA and timer, and external interrupts from IRQ and NMI.

Power dissipation of the chip with outputs open is from 350 mW to 610 mW. The operating voltage is from 3 to 3.6 V. The ambient temperature range is 0 to 70°C.

The chip is 27x27x2.15 mm in size. However, it has 256 pins housed in a Plastic Ball Grid Array (PBGA) package. The special requirements in handling BGA devices make it difficult to use such chips at CPU level. In view of this, Axis Communications produces development boards or systems which come complete with all the necessary peripheral devices and connectors.

## 3.2 ETRAX 100LX MCM 2+8

This is the true system-on-chip version produced by Axis Communications. It boasts a full Linux computer on a single chip. This is made possible by the incorporation of 2 MB Flash memory and 8 MB SRAM. The Multi-Chip Module (MCM) uses High Density Packaging (HDP) technology that integrate naked dies (i.e. chips without their capsules, e.g. ETRAX, RAM, Flash) and other components (like resistors and capacitors) to provide very compact overall module. It is built around the ETRAX 100LX system-on-chip processor. All necessary components for building a networked embedded system are included. In a 256-pin PBGA package measuring only 27x27 mm there are these components:

- ETRAX 100LX System-On-Chip
- 2Mbyte of Flash memory
- 8Mbyte of SDRAM memory
- Ethernet transceiver
- Reset circuitry
- ~50 passive components (resistors and capacitors)

The only mandatory external components are a 3.3 V power supply and a 20 MHz crystal oscillator. Additional flash memory and SRAM are not mandatory but may be added externally to the module.

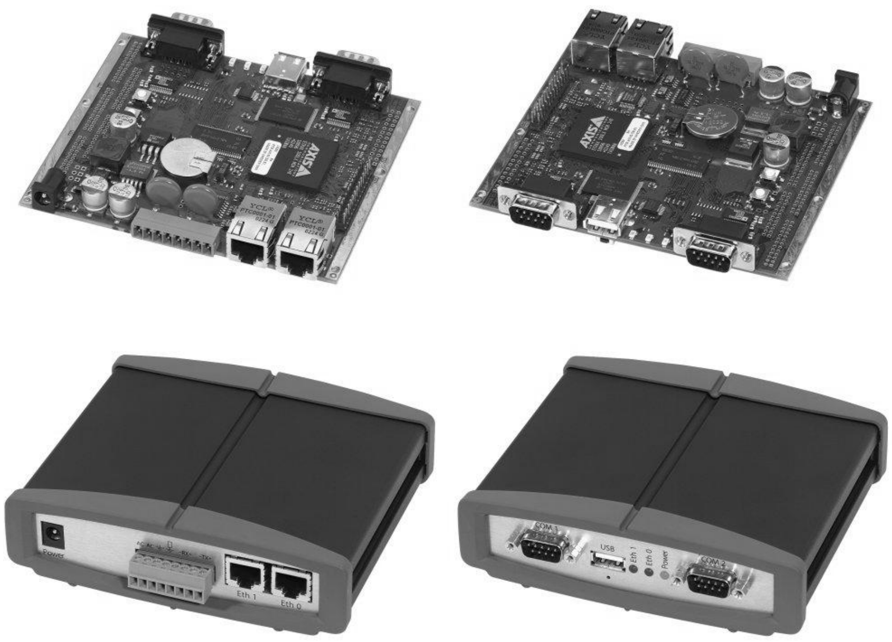
Due to the additional components, power consumption increases to 900 mW (typical) or 1100 mW (maximum). This module is priced at $75.

## 3.3 Axis 82 - Development Board

As mentioned earlier, the ETRAX 100LX or the MCM, while having practical all that's required to form a complete system, the PBGA package make it impossible for most people to access the I/O! Axis 82 is the solution to this problem. It is a development board that has all the hardware including indicators, connectors and even a casing as in case of Axis 83. However, an equally important additional feature is the inclusion of more Flash memory and SRAM. This is needed because the on-chip 2MB + 8MB memories are really bare bone minimum for the Linux environment. There is hardly any memory left for user applications. The main features of this 140x118 mm board are as follows:

ETRAX 100LX MCM 2+8
FLASH: 6 MByte (2MB inside the MCM chip)
RAM: 16 MByte SDRAM, 32-bit interface (8MB inside the MCM chip)
DUAL Ethernet 10/100 MBit Twisted Pair
One using the internal full-speed 100MBit controller in the ETRAX
One using a Ethernet-to-USB controller, thus limiting the theoretical maximum speed to 12MBit (i.e. the max speed of USB 1.1)
RS-232 serial ports, 9 pin male D-SUB
(with RX, TX, RTS, CTS, DTR, DSR, CD and RI signals)
1 RS-485/RS-422 serial port on a screw terminal block
(1 combined RX/TX pair and 1 TX pair)
Power LED, Eth0 LED, Eth1 LED
TEST button
RESET button
BOOT button - to enable network boot
Power: 9-24 V AC (or DC) on a standard connector and on screw terminal block
1 parallel port on pin headers (3.3 V, with buffers)
Full access to system bus and other important pins on ETRAX
Shipped with Axis embedded Linux port

This Axis 82 development board is priced at $299 while the Axis 83 which has a casing is at $309.



## 3.4 ETRAX 100LX Software and Development Tools

There is comprehensive software development support offered by the manufacture at the following website http://developer.axis.com/doc/index.html. The Linux kernel 2.4 for ETRAX 100LX including device drivers for all ETRAX ports and the GNU CC including compiler, debugger and other tools, are available for download.

The SDK includes a number useful applications. Many are standard applications compiled for the ETRAX platform, but some are specifically developed for the ETRAX platform. They are tabulated below:

| | |
|---|---|
| **Init** | Implements a mini init daemon. |
| **Telnetd** | Simple Telnet server - allows you to log on to the developer board. |
| **Inetd** | This program invokes all internet services as needed. |
| **in.telnetd** | Telnet server. This is the default telnet server. |
| **Sftpd** | Lightweight FTP server. |
| **Sftpclient** | Lightweight FTP client. |
| **shells/ash** | Ash is a fairly small Bourne compatible shell which offers the power and flexibility of shell scripting for an |

| | |
|---|---|
| | embedded system. |
| **Boa** | High performance web server with support for CGI. |
| **Smtpclient** | This program is a minimal SMTP client that takes an email message body and passes it on to a SMTP server (default is the MTA on the local host). |
| **Bootblocktool** | Sets/gets bootblock parameters using a specified device. |
| **hwtest/IO** | Input / Output test utility. |
| **hwtest/serial** | Program to test serial port functionality. |
| **hwtest/hwtest** | Program for setting or getting status from parallel port serial port, button, and some other useful devices. |
| **sysklogd** | Sysklogd provides two system utilities which provide support for system logging and kernel message trapping. Support of both internet and Unix domain sockets enables this utility package to support both local and remote logging. |
| **editors/easyedit** | An easy to use text editor. |
| **editors/editcgi** | This is a simple CGI based editor and file browser. |
| **login** | Login is used when signing onto a system. It can also be used to switch from one user to another at any time (most modern shells have support for this feature built into them, however). |
| **dhcp** | Complete DHCP client. |
| **busybox** | Busybox combines tiny versions of many common UNIX utilities into a single small executable. |
| **ipsetd** | Daemon for setting IP address with ARP+Ping in user-space. |
| **iptables** | Implements a NAT (Network Address Translation) firewall. |
| **tools/gdbserver** | The gdbserver will help you debug user applications running on the Developer Board for Bluetooth |
| **utils/eraseflash** | Flash erasing utility. |
| **utils/readbits** | Utility for reading bits on ETRAX general purpose I/O (GPIO) ports. |
| **ppp-2.4** | PPP daemon, which negotiates with the peer to establish the link and sets up the PPP network interface. |

# 4 PC/104 & PC/104-Plus

Hardware used in embedded systems is by no means limited to the very small microcontrollers and highly integrated system-on-chip modules introduced in the previous lectures. In many embedded systems, PCs are used as embedded processors. It is the application rather than the hardware that defines embedded system processors.

PCs – whether the desktop, notebook, tablet or even the pocket version – are indeed widely used in embedded systems, especially in laboratories. It is often simpler or even cheaper to use the standard PC as an embedded processor because of the accessibility, software support and availability of a host of peripheral devices. However, there are shortcomings too. Amongst them are size, reliability and possibly cost.

In view of these disadvantages of standard PCs as embedded processors, many manufacturers have come out with a myriad of compatibles, mostly in the form of single board computers (SBCs). They are typically PCs shrunk into small packages, offering more or less the power of the PC together with standard peripherals, and capable of running most PC software. There are hundreds if not thousands of different SBCs in the market catering for a very wide range of embedded system applications. Some are specifically designed to operate in very harsh industrial environments; others offer a cheaper solution than the standard PCs. Yet others have small form factors and are lightweight.

One family of small form PC compatible embedded processors that is gaining market share is the PC/104 and PC/104 Plus. In 1992, an industry wide effort led to the setting up of a PC/104 Consortium consisting of 12 companies, which then produced the standard. Now over 100 companies are members of the Consortium. The initial standard was to offer an open design that had the power and flexibility of the PC but small in size and suitable for embedded system applications.

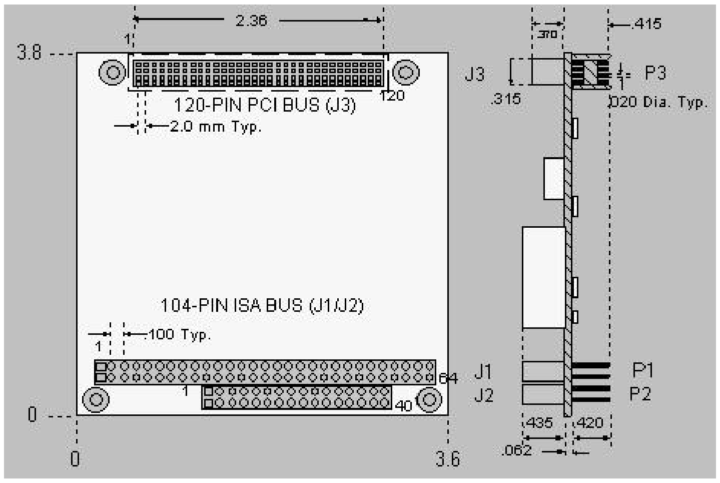
PC/104 technology is based on the ISA bus that was used in PCs. In fact, the 104 in the name is derived from the number of conductors used in the ISA bus. This 16-bit bus system is really very powerful, and is more than sufficient for most embedded system applications. Thus despite the fact that the IEEE P996 ISA Bus Specification has been withdrawn and major PC manufacturers have stopped manufacturing ISA bus systems, the PC/104 continues to flourish in the embedded system world.

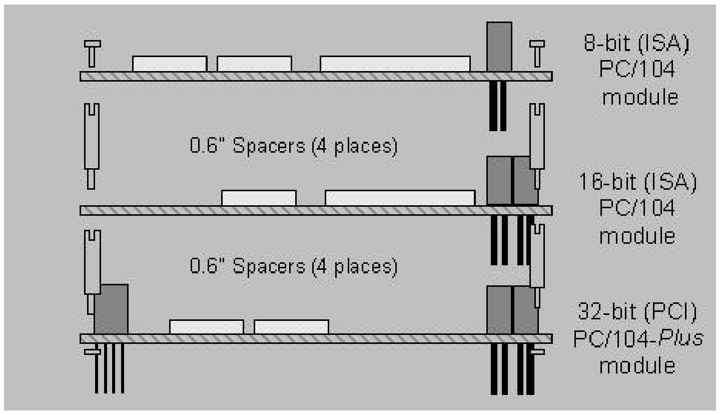
In order to tap the power of the higher speed, 32-bit PCI bus, the Consortium added the PC/104-Plus standard in 1997. However, this does not replace but merely complements the PC/104.

## 4.1 PC/104 and PC/104-Plus Standards

In essence, the PC/104 standards are self stacking modules with 104-pin ISA or 120-pin PCI bus connectors that allow multiple modules to be added without using backplanes or motherboard. Each module is only 3.6 x 3.8 in size and power consumption is typically 1 to 2 watts per module. PC/104 modules can also be arranged on a big custom carrier board instead of being stacked. In such cases application specific interfaces can be connected to various modules easily.

PC/104-Plus is compatible with PC/104 and supports 32-bit PCI interconnection. An additional 120-pin small pitch 2-mm connector is included on the other side of the ISA connector. Data throughput can now reach 132 MBytes/second which is 26 times that of PC/104. Bus drive current is 3 mA minimum while loading is 700 uA maximum. Five modules including the base CPU module can be stacked. The following diagrams show the physical dimensions and the stacking arrangement.





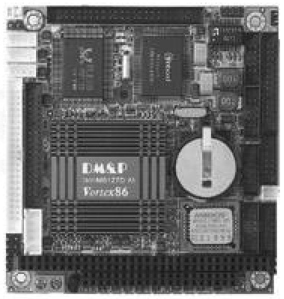
Technical specifications of PC/104 and PC104-Plus can be downloaded from the PC/104 Consortium website:
http://www.pc104.org/technology/pc104_tech.html

Since the functional specifications are really based on ISA and PCI standards, these two open standards should also be referred to.

## 4.2 PC/104 & PC/104-Plus Products

Almost every SBC manufacturer is producing PC/104 and PC/104-Plus products, including CPU and most peripheral modules. In the case of PC/104-Plus, high performance SBC, full motion video modules, high speed LANs, interfaces including 100BaseT, USB, IEEE-1394, high speed data acquisition, etc. are either available or being developed.

As a reference, one PC/104 SBC, Vortex86-6071, from ICOP Technology Inc., Taiwan is outlined below:



This is an embedded Vortex86 166 MHz system-on-chip CPU module with a lot peripherals included as can be seen from the manufacturer's specifications:

| | |
|---|---|
| **Chipset** | Embedded Vortex86, 166MHz System-on-Chip CPU<br>Real Time Clock with lithium battery backup |
| **BIOS** | AMI BIOS |
| **RAM** | 128MB SDRAM on board |
| **I/O** | Enhanced IDE interface x 1<br>RS232 port x 3<br>RS232/RS485 port x 1<br>Parallel Port x 1<br>FDD interface x 1<br>USB ports x 2 |
| **Bus Interface** | ISA and PC/104 standard compliant |
| **Connector** | 2.0mm 44-pin box header for IDE<br>2.0mm 10-pin box header x 4<br>2-pin header for RS-485<br>2.0mm 26-pin box header<br>2.0mm 34-pin box header<br>2.54mm USB ports x 2<br>*Audio connectors:*<br>4-pin header for Mic-in,4-pin header for line-in,4-pin header for line-out<br>10-pin box header for VGA connection<br>44-pin box header for LCD connection<br>5-pin box header for AT-keyboard<br>5-pin box header for PS-2 Mouse<br>RJ-45 connector for 10/100Base-T |
| **Display** | SiS550 embedded SOC, AGP Rev 2.0 Compliant<br>Shared system memory area up to 128MB<br>CRT/LCD display |

| | |
|---|---|
| | Resolution up to 1920 x1440 true colors |
| **LAN** | Realtek 8100B single chip<br>Full-duplex transfer mode, doubles effective band-width<br>NE2000 compatible with built-in 16KB RAM buffer<br>Throughput: 10/100 Mbps |
| **Audio** | Full compliant with AC97 CODEC V2.1<br>Internal MIC-in, Line-in and Line-out Interface |
| **Power Require-ments** | Single Voltage +5V @1.4A |
| **Dimensions** | 90(L) x 96(W) mm (3.54" x 3.78") |
| **Watch Dog Timer** | Software Watchdog Timer<br>Three 8254 Compatible Programmable 16-bit Count-ers |
| **Operating Tem-perature** | -20~+60°C |

# 5 Embedded Systems Networking in Distributed Environment

## 5.1 Introduction

There are two broad methods of connecting embedded systems in a distributed environment:

(1) Non-Internet connection
(2) Internet connection

Before the proliferation of the Ethernet and Internet, connecting embedded systems in a distributed environment such as a laboratory or a factory was usually achieved by conventional parallel or serial connections. Parallel connections were suitable for short-range use, typically within a few meters, limited by difficulties in driving synchronous multiple signals and the cost of cabling. The advantage of parallel connection was obviously higher data rates on a parallel bus. Parallel bus standards at Mbps rates were implemented when the standard serial link was still at Kbps.

While there are dedicated parallel bus standards designed for specific industries or environments such as the General Purpose Interface Bus (GPIB), the Computer Automated Measurement And Control (CAMAC) and the Small Computer System Interface (SCSI), the most common method is the parallel port, used typically for printer connection. This is simple to use and is readily available, being a standard I/O port in all PCs. The newer version (IEEE 1284) available in most new PCs is being used for many purposes other than connecting to a printer.

However, by far the most important networking technique in a distributed environment beyond the size of a test bench is, of course, the serial bus, exemplified by the RS-232-C (Recommended Standard number 232, revision C from the Electronic Industry Association) and its related standards. Like the parallel port, the use of this serial technique is extremely handy. There are two serial ports available in most standard desktop PCs – COM1 and COM2. Users and manufacturers alike are taking advantage of these two serial ports for almost every type of equipment.

The limitations of the standard PC serial ports are data rate and distance, being 20 Kbps and <50 feet. Related standards such as RS-485 eliminate such limitations to a great extent while still maintaining the simplicity of the standard. For example, RS-485 is capable of 4000 feet and 10 Mbps performance. It has an added advantage of a bus configuration which allows it to connect to 32 transmitters and 32 receivers. (For comparison, the RS-232 is a point-to-point connection which can only connect two machines together.) Although existing OS in PCs would not be able to handle continuous 10 Mbps data rate, the 4000-foot transmission distance is a welcome feature in a distributed laboratory environment.

As newer equipment and gadgets (especially video devices) demand higher and higher data rates, there are a couple of high-speed serial communication standards that have been introduced to the PC world in recent years. Universal Serial Bus (USB) provides for peripheral speeds of up to 1.5 Mbps for low-speed devices and up to 12 Mbps for full-speed devices. The latest USB 2 specification increases the data rate to 480 Mbps. The cable length is limited to 5 metres, extendable to 25 metres using 4 hubs or with active cables. A total of 127 devices may be connected.

A competing standard to USB is the IEEE 1394 (FireWire or iLink) which has a maximum data rate of 400 Mbps and is cable of connecting up to 63 devices. At this dazzling speed, full frame rate video can be transmitted from cameras to PCs directly through a thin cable. The cable length is 4.5 metres, again extendable with hubs and repeaters.

Some applications call for wireless connection. This is provided by IrDA, a standard defined by the Infrared Data Association. It specifies wireless transfer via infrared radiation at 875 nm. IrDA 1.1 has a maximum data rate of 1.152 Mbps. At this rate, the range is small, typically less than 20 cm.

The recent ubiquitous deployment of the Ethernet and Internet, together with the availability of low cost embedded systems capable of functioning as web servers, has changed everything in networking equipments in a distributed environment. It is now the age of the Internet! In this respect there are two approaches. One is to forget your old equipment and design new web-based ones and network them. The other is to add a web-enabled front-end or intermediary to your existing equipment and thus make it accessible via the Internet. Prudence and reality tell us that there are many situations where existing equipment cannot be abandoned or redesigned. Thus the second approach is an important one. Indeed, many companies dedicated to providing such hardware and software have appeared in the last couple of years. Their solutions are typically very attractive and cost effective, and we shall look at a couple of them.

If you embark on an entirely new project then you have two options in hardware for your embedded systems. If you are lucky enough to have a large budget, use a PC or one of its derivatives (SBC and embedded PC) as your embedded processor. The interface hardware can also be purchased as standard plug-in boards on the PCI (Peripheral Component Interconnect) bus. This eliminates hardware design completely. There are numerous suppliers producing a plethora of I/O boards. You can almost always find what you want provided you can afford it. We shall not look into this area in these lectures.

Sometimes a PC is an overkill for your applications or your budget simply cannot afford using PCs. Until recently, building web-based applications under such situations has been difficult. One often settled for the non-Internet option. Rather affordable hardware is currently available in the market either as components or ready built modules that are web-enabled. We shall look into this area.

## 5.2 Non-Internet Connection

The hardware or physical layer specifications of a few common methods of connecting embedded systems are described in this section. They are

(1) The parallel port
(2) The serial port
(3) Selected high-speed ports

## 5.3 The Parallel Port

Originally intended for printer connection, this port has undergone several revisions and it now provides bi-directional connection to a host of equipment. The initial printer port in the IBM PC in 1981 was really designed for unidirectional connection to the relatively slow dot-matrix printers at the time. It transferred data at 150 kilobytes/second and was software intensive. Lack of design standards forced a cable limitation of 6 feet. In 1987, IBM PS/2 was introduced and this PC enhanced the parallel port by adding bi-directional data flow.

Printer and computer manufacturers started to enhance the parallel port standard in 1991 and this resulted in two improved standards: EPP (Enhanced Parallel Port) and ECP (Extended Capability Port). EPP was introduced by Intel, Xircom and Zenith Data Systems and was used primarily by non-printer peripherals such as CD-ROMs, tape drives, hard disks etc. ECP was introduced by Hewlett Packard and Microsoft and was used primarily by new printers and scanners.

In 1994 the current IEEE 1284 standard, “Standard Signalling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers”, was released. This standard encompasses all the other modes of parallel port, viz.

(1) Compatibility mode – Centronics or standard mode
(2) Nibble mode – 4 bits at a time using status line for data
(3) Byte mode – 8 bits at a time using data line
(4) EPP – bi-directional, up to 2 Mbytes/second
(5) ECP – bi-directional, DMA transfer, up to 2 Mbytes/second

The standard specifies a cable length of 10 metres.

The original parallel port consists of 17 signal lines falling into three groups – Data (8 lines), Status (5 lines) and Control (4 lines). The remaining 8 lines in the DB25 connector port are grounded. For compatibility, the 17 signal lines are maintained even in the advanced EPP and ECP mode. However, the function and designation of the lines are modified according to the new requirements of the modes.

In the original PC, the data lines are output-only lines and thus they cannot be used to read data from peripheral devices. (The nOC output control pins of the 74LS374 D-type flip-flops are permanently grounded to enable all output lines.) To get round this problem, the Nibble mode uses 4 of the 5 status (input) lines to read 4 bits at a time from the peripheral devices. While this is more complicated and slow, it has the advantage of compatibility with the oldest PCs.

Most PCs produced after 1987 have the ability to control the direction of the data port thus making bi-directional transfer of 8-bit data using the same port possible. This is the Byte mode.

For the first three modes of communication – Compatibility, Nibble and Byte – handshaking of data between the PC and peripheral is carried out by the CPU and it takes several clock cycles to transfer a single byte or nibble. This limits the data rate to 150K bytes per second for byte-wide transfer and to 50K bytes per second for nibble-wide transfer. These data rates are however quite respectable for many laboratory applications. Because of their simplicity, they are used in many embedded system applications. Hence, we shall look into the actual transfer cycles in these modes.

The EPP mode enhances the performance of the parallel port by increasing the data rate to 2M bytes per second while still maintaining compatibility with the standard port. A read or write cycle is done within one ISA I/O cycle. This is achieved with a local protocol between the host and peripheral using a state machine and the necessary hardware to carry out interlocking handshakes. To enhance the performance, the EPP protocol defines four types of data transfer cycles: (1) data write, (2) data read, (3) address write and (4) address read.

While data cycles and address cycles are intended for transferring data and address/command respectively, they are really just two transfer channels and can be used flexibly by device manufacturers.

The EPP data rates are sufficient for use with network adapters, hard disks, other tape storage devices and CD ROMs. Thus many device manufacturers adopted the EPP mode swiftly as an optional data transfer method.

The ECP came even later as another high performance parallel port mode. It is specifically designed with printers or similar devices in mind. The protocol allows for (1) data and (2) command cycles in both forward (write) and reverse (read) directions. In this respect it is similar to the EPP. However, it includes several features that are very specific to implementations but powerful. They are (1) Run Length Encoding (RLE) data compression, (2) FIFOs on both forward and reverse channels and (3) DMA and programmed I/O for the host register interface.

The RLE, with a compression ration up to 64:1, is useful for printers and scanners that handle raster images with large amounts of identical data (blank or black). The channel addressing concept is intended for addressing multiple logical devices using a single physical port. A good example of such a requirement is the combined Fax/Scanner/Printer machine.

The following tables show the pin designations and other relevant information for the first three modes of operation. These are modes that are widely used by embedded systems implementers in the laboratory. The remaining two modes, EPP and ECP, while more advanced, are not as commonly used in simple laboratory applications. When high data rates are required, it is now advisable to implement high-speed serial link interfaces such as USB or FireWire.

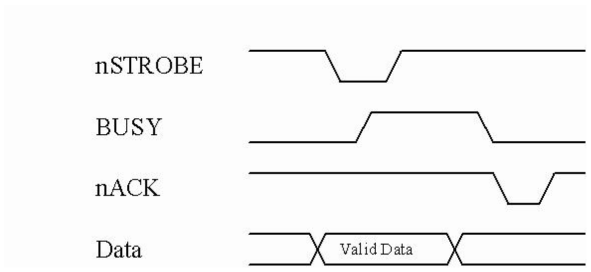
Compatibility Mode (or Standard Parallel Port Mode)

| Pin | Signal | I/O | Reg | Description |
|---|---|---|---|---|
| 1 | nSTROBE | Out | CR-0 | Falling edge strobes data byte into printer |
| 2 | D0 | Out | DR-0 | Bit 0 of data |
| 3 | D1 | Out | DR-1 | Bit 1 of data |
| 4 | D2 | Out | DR-2 | Bit 2 of data |
| 5 | D3 | Out | DR-3 | Bit 3 of data |
| 6 | D4 | Out | DR-4 | Bit 4 of data |
| 7 | D5 | Out | DR-5 | Bit 5 of data |
| 8 | D6 | Out | DR-6 | Bit 6 of data |
| 9 | D7 | Out | DR-7 | Bit 7 of data |
| 10 | nACK | In | SR-6 | Low pulse indicates byte received |

| | | | | |
|---|---|---|---|---|
| 11 | BUSY | In | SR-7 | High indicates printer busy |
| 12 | PE | In | SR-5 | High indicates out of paper |
| 13 | SELECTED | In | SR-4 | High indicates printer online |
| 14 | nAUTOFEED | Out | CR-1 | Low to insert line-feed on each carriage return |
| 15 | nERROR | In | SR-3 | Low to indicate printer error |
| 16 | nINIT | Out | CR-2 | Low to reset printer |
| 17 | nSELECT | Out | CR-3 | Low to select printer |
| 18-25 | Ground | | | Ground |

CR - Control Register
DR - Data Register
SR - Status Register

## Compatibility Mode Data Transfer Cycle

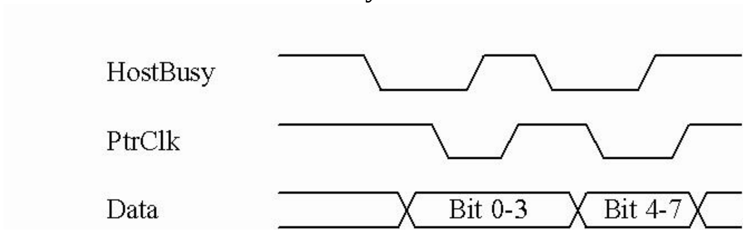


## Nibble Mode (Reverse Transfer)

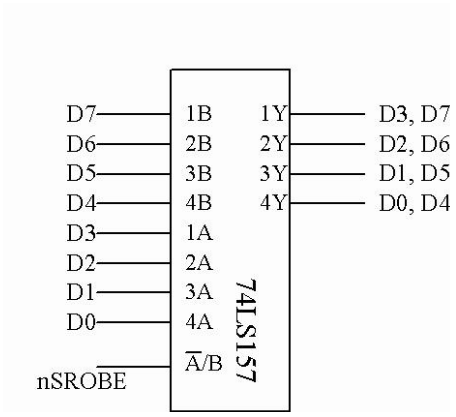
| Pin | SPP Signal | Nibble Mode | I/O | Description |
|---|---|---|---|---|
| 1 | nSTROBE | Not used | Out | Not used |
| 2 | D0 | Not used | Out | Not used |
| 3 | D1 | | Out | |
| 4 | D2 | | Out | |
| 5 | D3 | | Out | |
| 6 | D4 | | Out | |
| 7 | D5 | | Out | |
| 8 | D6 | | Out | |
| 9 | D7 | | Out | |
| 10 | nACK | PtrClk | In | Low indicates valid nibble<br>High in response to HostBusy going high |
| 11 | BUSY | PtrBusy | In | Data bit 3 then bit 7 |
| 12 | PE | AckDataReq | In | Data bit 2 then bit 6 |
| 13 | SELECTED | Xflag | In | Data bit 1 then bit 5 |

| | | | | |
|---|---|---|---|---|
| 14 | nAUTOFEED | HostBusy | Out | Low indicates host ready for nibble<br>High indicates nibble received |
| 15 | nERROR | nDataAvail | In | Data bit 0 then bit 4 |
| 16 | nINIT | NINIT | Out | Not used |
| 17 | nSELECT | 1284Active | Out | High indicates 1284 mode |
| 18-25 | Ground | Ground | | Ground |

Nibble Mode Data Transfer Cycle



On the peripheral side, the reverse transfer in the Nibble mode can be implemented with a quad 2-line-to-1 multiplexer such as the 74LS157 as shown in the following diagram.

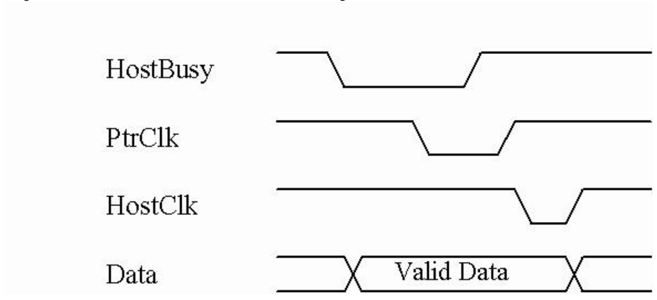


Byte Mode

| Pin | SPP Signal | Byte Mode | I/O | Description |
|---|---|---|---|---|
| 1 | nSTROBE | HostClk | Out | Pulse low to indicate byte received |
| 2 | D0 | D0 | I/O | Bit 0 from peripheral to host |
| 3 | D1 | D1 | I/O | Bit 1 from peripheral to host |
| 4 | D2 | D2 | I/O | Bit 2 from peripheral to host |
| 5 | D3 | D3 | I/O | Bit 3 from peripheral to host |
| 6 | D4 | D4 | I/O | Bit 4 from peripheral to host |
| 7 | D5 | D5 | I/O | Bit 5 from peripheral to host |

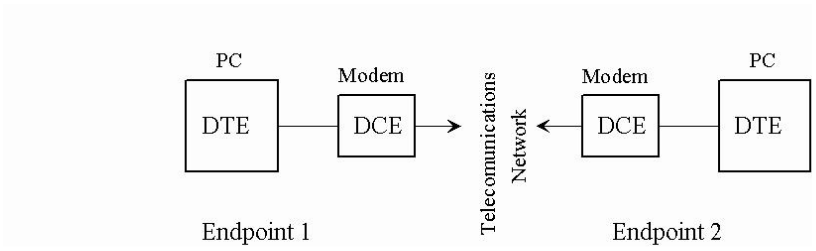
| | | | | |
|---|---|---|---|---|
| 8 | D6 | D6 | I/O | Bit 6 from peripheral to host |
| 9 | D7 | D7 | I/O | Bit 7 from peripheral to host |
| 10 | nACK | PtrClk | In | Low indicates valid data<br>High in response to HostBusy going high |
| 11 | BUSY | PtrBusy | In | Printer busy |
| 12 | PE | AckDataReq | In | Follows nDataAvail |
| 13 | SELECTED | Xflag | In | Extensibility flag, not used |
| 14 | nAUTOFEED | HostBusy | Out | Low indicates host ready for nibble<br>High indicates nibble received |
| 15 | nERROR | nDataAvail | In | Low indicates data available |
| 16 | nINIT | NINIT | Out | Not used |
| 17 | nSELECT | 1284Active | Out | High indicates 1284 mode |
| 18-25 | Ground | Ground | | Ground |

Byte Mode Data Transfer Cycle



## 5.4 The Serial Ports

The two serial communication ports in the PC follow the RS-232 standard of the Electronics Industry Association. It was introduced in early 1960s and has since been the most widely used serial communication technique in the laboratory. The title of the standard, "Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange" indicates the original intention. That is, to connect Data Terminal Equipment (DTE) to Data Communications Equipment (DCE). DTE and DCE are the two pieces of equipment at the two endpoints of a telecommunication channel. The DTEs used to be dumb terminals with which the users interact. The DCEs were modems that connected to telephone lines. In most laboratory and office environments now, the DTEs are more likely to be PCs. The DCEs may still be modems, but they may also be non-existent because the other endpoint is nearby and it is easier to use straight-through cables instead of the telephone network. In this case, the DCEs are called Null Modems.

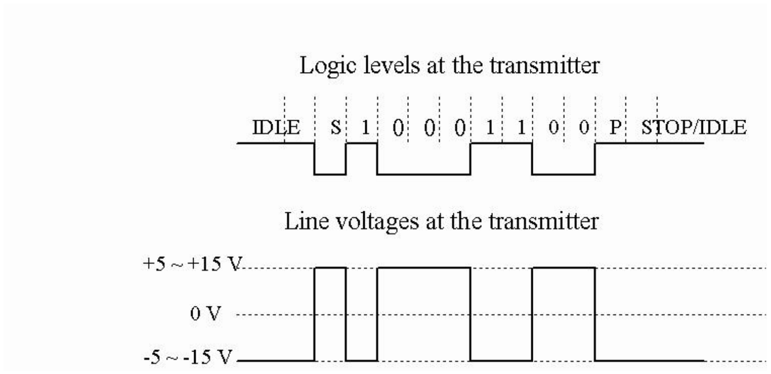


The RS-232 uses an asynchronous mode. Typically, data is transferred in symbols or words of 5 to 8 bits, one bit at a time, least significant bit first, at a predetermined speed called the baud rate. However, from one word to the next, the time interval is not fixed and thus asynchronous. A long string of words may be transferred in a contiguous manner, one immediately trailing the previous one, or they may be sent intermittently, with varying pauses between words.

No clock or other timing signal is sent. It thus requires a start bit to indicate the beginning of a byte and a stop bit to signify the end. It is the receiver's job to sample the bits based on these marking bits. For the purpose of error checking, a parity bit (either even or odd parity) may be added at the end of each word or symbol. The transmitter computes the parity bit based on all the data bits. The receiver checks if the actual parity bit value corresponds to the calculated value.

The actual signals sent on the line are positive and negative voltages with respect to ground potential. A bit 0 or logical 0 is sent as SPACE or +5 to +15 volts; a bit 1 or logical 1 is sent as MARK or –5 to –15 volts. The receiver recognises –3 to –25 volts as MARK and +3 to +25 as SPACE. The region between –3 volts to +3 volts is a transition region where the state is undefined. The wide range of voltages for the two valid states are very useful in actual implementation, taking care of line drop, supply fluctuation, ground potential difference etc. A number of restrictions imposed by the standard further enhance the robustness. For example, the output of the transmitter may be shorted to other lines without causing any damage to itself and the receiver must be able to withstand voltages in the range of –25 volts to 25 volts.

When a line is idle, it is in the MARK state. The start bit is SPACE, stop bit(s) is MARK. The timing diagram of sending numeral "1" or 0x31 with even parity is shown below.


Logic levels at the transmitter
IDLE
S
1
0
0
0
1
1
0
0
P
STOP/IDLE
Line voltages at the transmitter
+5 ~ +15 V
0 V
-5 ~ -15 V

The standard specifies a total of 21 control signals or 'circuits' on top of the two data lines (Transmit Data and Receive Data). The original specification thus requires a DB-25 connector with 25 circuits. The control lines are used to connect to a DCE or modem for control or test functions. It turns out that many of the control lines are not needed for standard modem connections. In fact they may not be used at all if an RS-232 port is connected to another RS-232 port on two pieces of equipment. In this case, the only requirement is of course to connect the Transmit Data of one DTE (the PC) to the Receive Data of the other. This crossing of two wires, from pin 2 to pin 3, constitutes a Null modem.

In most PCs, a subset of the lines is used for the serial port. Six control signals are chosen, making it possible to house the port in a DB-9 connector with 9 circuits. The connector on the PC or DTE is a male (or plug) while the corresponding one on the modem or DCE is a female (or socket). If two PCs are connected using the serial COM ports, the cable (or the Null modem) obviously has two female DB-9 connectors.

Three types of connectors are used for RS-232: DB-9 (9-pin D-shell connector), DB-25 (25-pin D-shell connector) and RJ45 (modular telephone connector). The various signal assignments on the DTE (or PC) side are as follows:

| **Name** | **Direction** | **Description** | **DB-9** | **DB-25** | **RJ45** |
|---|---|---|---|---|---|
| CD | ← | Carrier Detect | 1 | 8 | 2 |
| RxD | ← | Receive Data | 2 | 3 | 5 |
| TxD | → | Transmit Data | 3 | 2 | 6 |
| DTR | → | Data Terminal Ready | 4 | 20 | 3 |
| Gnd | | Signal Ground | 5 | 7 | 4 |
| DSR | ← | Data Set Ready | 6 | 6 | (1) |
| RTS | → | Request To Send | 7 | 4 | 8 |
| CTS | ← | Clear To Send | 8 | 5 | 7 |

| | | | | | |
|---|---|---|---|---|---|
| RI | ← | Ring Indicator | 9 | 22 | 1 |

Direction is DTE (PC) relative to DCE (modem)

The common bit rates of RS-232 are at 1200, 2400, 4800, 9600, 19200 bps. With modern drivers, higher bit rates, while non-standard, are possible.

The standard specifies a cable length of 50 feet or 2500 pF in cable capacitance, at 20K bps. If cables with lower capacitance are used, it is possible to transmit at distances greater than 50 feet. For example the UTP Cat-5 cable has a capacitance of 17 pF/ft. Thus about 150 feet of cable may be used. Likewise, reducing the transmission speed increases the maximum cable length. Laboratory tests showed that a cable length of 500 feet is possible at 9600 bps.

Flow control in RS-232 if needed is carried out either in hardware or software to prevent buffer overflow on the receiver. In the hardware or the RTS/CTS flow control method, the transmitter asserts a Request To Send signal when it has data to transmit. The receiver asserts a Clear To Send signal only when it is ready to receive data. In the software or XON/XOFF control mode, the receiver tells the transmitter to send more data by sending an XON control character (0x11). When the receiver buffer is nearly full, it stops the transmitter by sending an XOFF control character (0x13). Both the XON and XOFF are sent as data and no hardware control signals are involved.

Very small embedded systems connected to the serial port may get their power from the serial port itself although this is not part of the standard. A serial port mouse is such an example. Since DTR and RTS are often not used as intended, they may act as a source of small power supply. These pins are typically designed to drive a load of 3~7 kOhm at 7 to 11 volts and they can be used to drive a regulator to produce regulated 5-volt output, provided that the load current is small (typically less than 10 mA, depending on the type of driver used in the port). An output pin from a standard 1488 RS-232 driver chip can supply up to 6.5 mA at 5 volts and that for the MAX232 is 5 mA. Devices operated in this manner are called pin powered devices.

## 5.5 Differential Drive Serial Communication Standards (RS422, RS485)

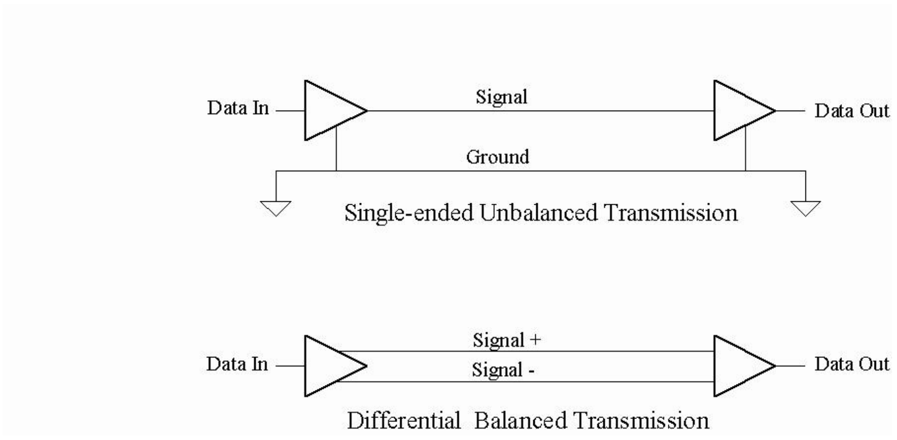
While RS-232 is simple and readily available, there are some shortcomings that reduce its speed and transmission distance. First, it assumes common ground potential between the two endpoints, the result of the single-ended (or unbalanced) drive. This may not be so when the separation of the devices increases.

A receiver at a different ground potential will sense the signal voltages with an undesired offset.

A single signal cable makes screening or shielding of electromagnetic noise difficult if not impossible. While a separate screen sheath may be used on the cable, internal noise generated by other signal lines is a problem. This crosstalk effect becomes more and more severe as transmission speed goes up.

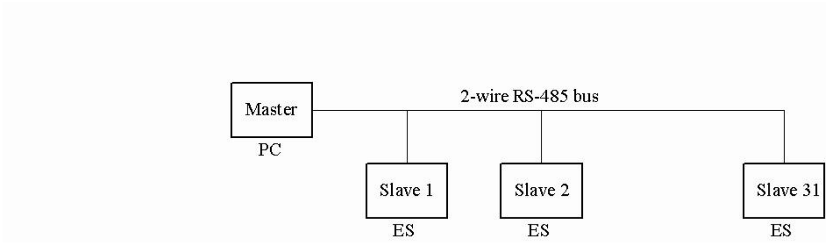
A different EIA standard, the RS-423 attempts to eliminate the shortcomings of RS-232 to a certain extent. By using different electrical parameters including a controlled slew rate on the signal (rise and fall times at 30% of unit interval at >= 1K baud), a speed of 100K bps at 4000 feet can be achieved.

However, a more significant enhancement of the serial communication technique is achieved by using differential or balanced drive. Two famous EIA standards that operate in this mode are the RS-422 and RS-485. A pair of wires is used for transmitting each signal. The voltage difference between the two wires is unaffected by difference in ground potentials and the receiver should function properly provided that the common mode voltage limit is not exceeded. If these two balanced wires are twisted together the difference voltage is unaffected by external electromagnetic noise, including crosstalk. Noises picked up by the two wires are likely to be identical and hence they are cancelled at the receiver input which is a differential voltage sensor.


Data In
Signal
Data Out
Ground
Single-ended Unbalanced Transmission
Data In
Signal +
Signal -
Data Out
Differential Balanced Transmission

Using this differential drive technique, transmission speeds up to 10Mbps can be achieved. The cable length is specified at 4000 feet. The RS-422 standard also specifies a multi-drop (party-line) arrangement where one driver (master) is connected to 10 receivers (slaves) that listen simultaneously. The master transmit lines are connected to all the receivers of the slaves. The slave transmit signal is carried on another pair. They are all connected to the receiver of the master. At any one time, only one of the slave transmitters is turned on to avoid garbling of data.

The RS-485 is a true multipoint arrangement whereby all the transmitters and receivers are connected to a single pair of wires, the bus. This is possible because the transmitter can be put into a high impedance or tri-state mode. Up to 32 pairs of transmitters and receivers may be connected together. The standard does not specify the method for orderly transmission in this bus system. One simple implementation method is for the master to initiate a communications request to a slave station by addressing it. It turns off its transmitter immediately and the slave can then transmit on the bus.



A 9-bit protocol is often used in the multidrop network shown above. An extra 9th bit is added to the typical 8-bit data in the serial transmission to indicate it is an address frame. When the master wants to transmit a block of data to a particular slave, it first sends an address byte as data and turns on the 9th bit. The particular slave, whose address matches that sent by the master, knows that it is being addressed. It thus receives the subsequent block of data based on whatever protocol was agreed upon earlier. Other slaves ignore the block of data not intended for them.

As standard transceivers (or UART) for RS-232 do not support 9th bit addressing, the parity bit is often used as the 9th bit. This is the case if the PC is used. In this case, the parity checking capability is lost of course. However, this is often not a problem as many existing applications ignore the parity checking in the first place. Other block checking techniques including the CRC are used.

A summary of the RS-232 and RS-485 electrical specifications is shown below.

RS-232

| **Parameter** | **Conditions** | **Min** | **Max** | **Units** |
|---|---|---|---|---|
| Output (Open Circuit) | | | 25 | V |

| | | | | |
|---|---|---|---|---|
| Output (Loaded) | 3 K <=$R_L$<= 7 K | 5 | 15 | V |
| Output Resistance | -2V<=Vo<=2V | | 300 | Ohm |
| Output Current (Short-Circuit) | | | 500 | mA |
| Output Slew Rate | | | 30 | V/µs |
| Maximum Load Capacitance | | | 2500 | pF |
| Receiver Input Resistance | 3V <= $V_{IN}$<=25V | 3000 | 7000 | Ohm |

RS-485

| **Parameter** | **Conditions** | **Min** | **Max** | **Units** |
|---|---|---|---|---|
| Output (Open Circuit) | | 1.5<br>-1.5 | 6<br>-6 | V<br>V |
| Output (Loaded) | $R_{LOAD}$ = 54 Ohms | 1.5<br>-1.5 | 5<br>-5 | V<br>V |
| Output Current (Short-Circuit) | Output to +12V or -7V | | ±250 | mA |
| Output Rise Time | $R_{LOAD}$ = 54 Ohms<br>$C_{LOAD}$ = 50 pF | | 30 | % (bit width) |
| Output (Common Mode) | $R_{LOAD}$ = 54 Ohms | -1 | 3 | V |
| Receiver Sensitivity | -7 <= $V_{cm}$ <= +12 | | ±200 | mV |
| Receiver (Common-Mode) | | -7 | +12 | V |
| Receiver Input Resistance | | 12K | | Ohm |

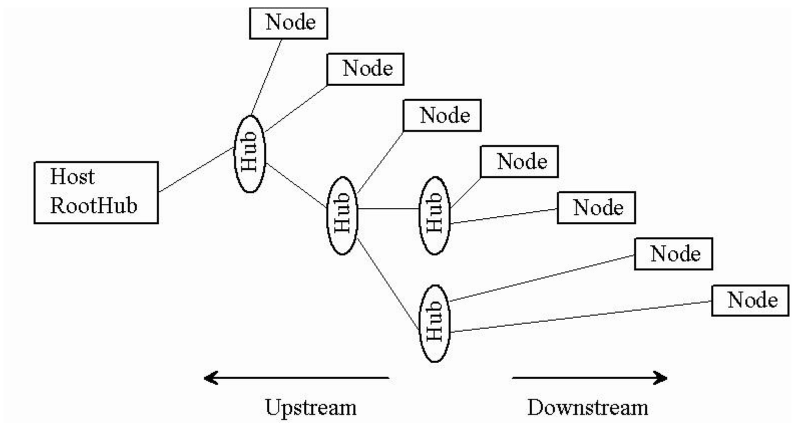
## 5.6 Universal Serial Bus (USB)

The Universal Serial Bus (USB) was standardised in 1995 by a group of companies to introduce an advanced serial bus to the PC. In the short period since its appearance, it has been widely accepted and all new PCs now incorporate USB ports. There are myriads of USB peripheral devices produced (over 1000 products have passed the compliance test), including mice, scanners, printers, digital cameras, hard disks, multimedia equipment, etc.

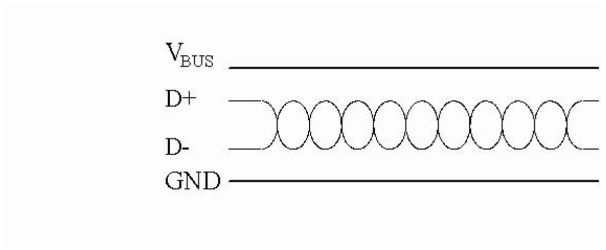
For embedded system developers, USB provides an alternative and more powerful way of interfacing to the PCs. It is a serial bus with hot-swap capability, meaning a device may be connected or disconnected without turning off the power.

USB 1.1 offers two data rates, full speed at 12 Mbps and low speed at 1.5 Mbps. USB 2.0 offers high speed at 480 Mbps. Devices of different speeds may be mixed and the bandwidth may be divided into asynchronous and isochronous streams. The latter offers guaranteed or reserved bandwidth and is useful for streaming devices, such as audio and video equipment.

A total of 127 devices may be connected to a single host. This is done by connecting multiple USB hubs in a daisy chain. Devices are connected to the downstream port of a hub while the upstream port is connected to the host or another hub (in the upstream of the bus topology). Devices are either self powered or powered from the bus. A typical USB hub may provide 100 mA on each downstream port in the Bus Powered mode, and 500 mA on the Self Powered mode.



USB devices are connected using a simple cable consisting of a twisted pair of wires for signals and an untwisted pair for power ($V_{bus}$ at 5 V and ground). Cable segments can be up to several metres long depending on the wire gauge used.



The clock speeds for high-, full- and low-speed operations are 480, 12, and 1.5 Mbps respectively. The voltage level for logical 1 at the final target connector is defined by (D+) - (D-) > 200 mV and that for logical 0 is (D-) - (D+) > 200 mV.

The USB uses a polled bus protocol whereby the host (or PC) initiates all data transfers. A transaction begins by the host sending a token packet. The packet contains information on the type and direction of transaction, device address, and endpoint number. Data transfer than occurs between the source and the destination by the source sending either a data packet or a packet indicating no data. The destination closes the transaction by returning a packet indicating whether the previous packet had been received successfully.

The pipe concept is used to refer to the channel between an endpoint in a device and that in a host. Two types of pipes are defined: stream and message. Streams have no defined data structure while messages have. A message pipe called Default Control Pipe is set up when a device is powered. It provides access to the device configuration, status and control information. Pipes have associations of data bandwidth, transfer service type, and endpoint characteristics like directions and buffer sizes. A flow control mechanism allows the construction of flexible schedules. This allows concurrent servicing of a heterogeneous mix of streams, i.e. multiple streams can be served at different intervals with packets of different sizes.

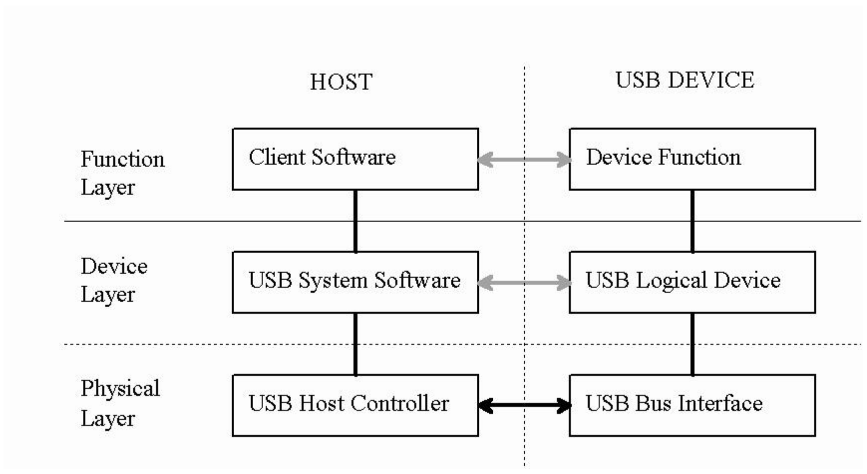
There are four types of data transfers: (1) Control Transfer, (2) Bulk Data Transfer, (3) Interrupt Data Transfer and (4) Isochronous Data Transfer.

Control transfer, a lossless data transfer method, is used to configure a device at attach time. Bulk data transfer is used for asynchronously transferring large volumes of data. Data integrity is ensured at hardware level by error detection and retries if necessary. An example is a transfer from the host to a printer.

Interrupt data transfer is used for transferring small amounts of data with small latency or rate specified by the device. Data are typically event notifications, characters or coordinates in one or more bytes. An example is the pointing device.

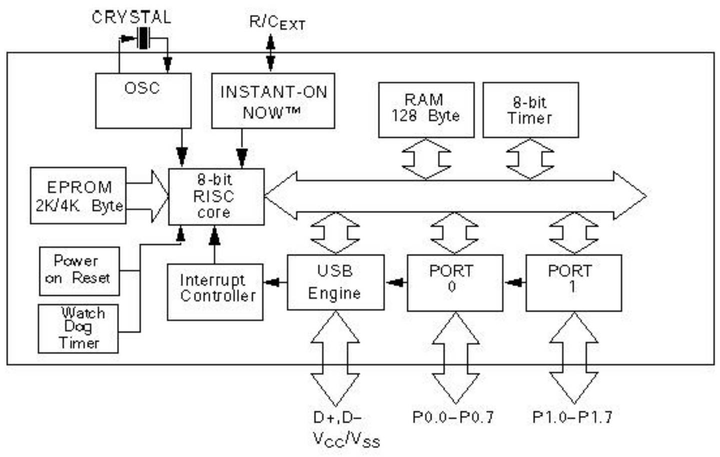
Isochronous data transfer is designed for real-time streams such as voice and video. A dedicated bandwidth is allocated for this transfer. Timeliness is ensured at the expense of occasional transient data losses. In other words, hardware retries are not used in case of data loss. This is known to be better for real-time voice and video transmission.

The USB standard implements a data flow model that can be divided intro three layers: (1) Function Layer, (2) USB Device Layer, and (3) Physical Layer as shown below.



The physical layer provides physical, signalling and packet connectivity functions between the host and a device. The device layer carries out general USB operations between the host and a device through a logical link. In the host, this is typically carried out in the operating system, as API for example. The function layer is used for high level function communication between the host and a device, again through a logical link as viewed by the software.

There are a number of manufacturers producing components specially designed for implementing embedded systems using USB. The functions of the physical and device layers are taken care of by USB chips. An example is the CY7C630/1XXA series produced by Cypress Semiconductor Corporation. This particular series of chips conforms to the USB 1.1 specifications operating at 1.5 Mbps. Members of the family have an 8-bit RISC microcontroller with a built-in 1.5-Mbps USB Serial Interface Engine (SIE). Memories are in the range of 128 bytes RAM and 2 to 4 Kbytes EPROM. Besides the USB interface, two 8-bit parallel ports are available. Such USB chips are suitable for small to medium scale applications including serial port conversion, keyboard, mouse, joystick and many others. A block diagram of a CY7C63000 chip is shown below.



## 5.7 IEEE 1394 Bus

This is an emerging high speed serial bus originally developed by Apple (called FireWire) and used by Sony in digital video equipment (called iLink). It was accepted as an industrial standard in 1995 (called IEEE 1394). Before the introduction of USB 2.0 specifications, the IEEE 1394 was the highest speed serial bus at 400 Mbps. 1.6 Gbps and 3.2 Gbps versions are being defined and developed. The following is a partial list of peripherals available with IEEE 1394 interface: digital video camcorders, digital cameras, printers, mass storage devices, ADCs.

The IEEE 1394 bus is similar to USB in principle. It has both asynchronous and isochronous modes of transmission suitable for both bursty data and real-time video streams. At 400 Mbps, full frame rate digital video streams can be transmitted. It has hot swapping capability. It allows up to 63 devices to be connected. Devices can also be bus powered or self powered.

One significant difference between IEEE 1394 and USB is the bus topology. The IEEE 1394 uses a peer-to-peer configuration, instead of the master-slave configuration. This means a PC is not needed to act as a host or master for communication. An IEEE 1394 digital video camera can be connected to a video player directly. Similarly, more than one PC can be connected to a digital camera to use the same video stream.

This, together with speed and other requirements, makes the specifications rather complex. The specifications document is a few hundred pages divided into many areas of applications, including instrumentation, industrial and automotive, on top of the more familiar video related specifications of digital video, MPEG, digital TV, etc. In the instrumentation area, one application is to replace the now rather slow IEEE 488 GPIB with an IEEE 1394 bus. There is an ongoing effort to define how the IEEE 488.2 command structure can be transmitted over IEEE 1394.

As in USB, IEEE 1394 uses a three layer model for communication: Physical, Link and Transaction. The physical layer handles the signals required by the bus; the link layer formats data into packets. Packets are passed on to the transaction layer which then presents them to the application above it. Semiconductor chips are available for handling the functions of the physical and link layers. Some of the transaction layer functions are also handled by the dedicated IEEE 1394 chip. The rest are in software, in the OS and the user programs.

In the Linux environment, version 2.2 onwards supports IEEE 1394. The subsystem has been included with the standard kernel sources since version 2.3.40. It is distributed as a patch to version 2.2. IEEE 1394 is also supported on the Windows and Macintosh platforms.

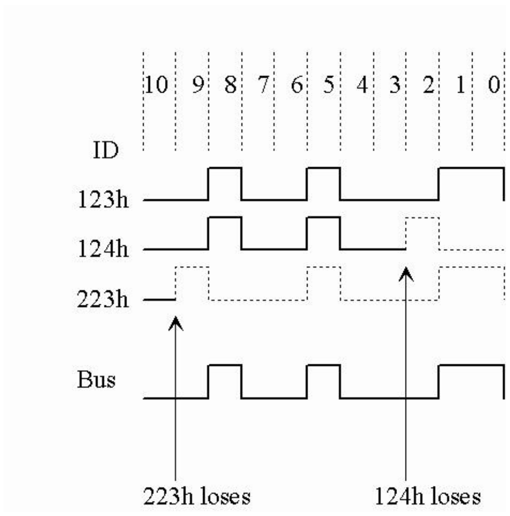
The IEEE 1394 cable consists of two differential signal pairs, a power and a ground line. Two types of connectors are defined: a six-wire version (measures 10 mm by 5 mm) and a four-wire version (measures 5 mm by 3 mm) without the power pair. The maximum cable length for speeds greater than 200 Mbps is 4.5 m. At lower speeds, cables up to 14 m long can be used. Multiple devices can be daisy-chained using repeaters to extend communication distance. It is interesting to note that the future IEEE 1394b specifications also support cable lengths of 100 m using plastic optical fibre and multiple kilometres using glass optical fibre. CAT-5 cables can be used at 100 Mbps at distances up to 100 m.

## 5.8 Controller Area Network (CAN)

The Control Area Network (CAN), originally developed for the automotive industry by Robert Bosch in 1980s, is a real-time serial data communications bus now being deployed in many distributed environment. It is now accepted as an international standard (ISO 11898 and ISO 11519 for high and low speed applications respectively). Many semiconductor manufacturers (e.g. Atmel, Fujitsu, Intel, Microchip, Motorola, Philips, Siemens, etc.) produce a wide range of controller and interface chips for CAN.

CAN controllers are physically small and relatively cheap devices used for real-time data acquisition and control applications. Data are transmitted on a two-wire bus using 5-V differential mode using a multicast protocol. A large number of devices may be connected to the bus. The limit is imposed by the drive capabilities of the control chips and up to 64 nodes is common. Unlike most other bus arrangements, devices are not identified by their device address. Data messages transmitted on the bus do not carry source or destination addresses. Unique message identifiers are used instead. A node sends out a message with an identifier and all the nodes on the bus receive it. It is up to the receiving node to decide whether to process the incoming message. The message identifier also serves as a priority indicator. The lower identifier has a higher priority.

Contention is solved by Carrier Sense, Multiple Access with Collision Detect (CSMA/CD) as in the Ethernet. However, an enhanced feature for non-destructive bitwise arbitration is used to resolve collision. This is achieved by a wired-AND mechanism. A logical 0 in the identifier bit pattern is considered dominant and it overwrites a logical 1 which is recessive as shown below. Thus a node sending out a message with an identifier 123h will overwrite a node with message identifier 223h and then with identifier 124h. Once a node transmitter with a lower priority loses, it turns into a receiver and listens to the message at a higher priority (or lower identifier value).



A useful feature in the message-based protocol of the CAN is the Remote Transmit Request (RTR). This allows a node to request information from other nodes. The protocol also permits additional nodes to be added without having to reprogram the existing nodes in the network.

Four different types of messages or frames are defined: (1) Data Frame, (2) Remote Frame, (3) Error Frame and (4) Overload Frame.

A data frame consists of the data field and several other fields to provide additional information. The whole frame can be divided into an Arbitration Field, a Control Field, a Data Field, a CRC Field, an Acknowledge Field and an End of Frame Field. The Arbitration Field is the message ID mentioned above and is used for prioritising messages on the bus. The prioritising field has 11 (standard frame) or 29 (extended frame) identifier bits and one RFR bit. If the RFR is set, the frame becomes a remote frame. The Control Field consists of a bit that signifies an extended frame and a four-bit Data Length Code (DLC). The value of the DLC is between 0 and 8 representing 0 to 64 bits. The remote frame has no data field, irrespective of the value of the DLC.

The CRC Field is a 15-bit CRC field with a delimiter. The Acknowledge Field is used to indicate if the message was received correctly, irrespective of whether it was processed or ignored. This is done by asserting a dominant bit on the bus at the ACK slot bit time.

When a node detects an error, it sends an error frame. When a node is not ready to receive additional messages, it sends an overload frame on to the bus. CAN was designed with error tolerance and robustness in mind. Thus, there are many types of errors being used: (1) CRC Error, (2) Acknowledge Error, (3) Form Error, (4) Bit Error, and (5) Stuff Error.

Although CAN is not a standard bus in PCs, CAN interface cards are readily available. As mentioned earlier, a wide range of CAN-based controller chips are available as standard semiconductor parts. There are two hardware implementations: (1) Basic and (2) Full. The former uses a standalone CAN controller connected to an existing microcontroller. The latter integrate a CAN controller, a CPU and a RAM in a single package.

Data rates of CAN bus depend on the length of the bus. For ISO11898 compliant devices, 1 Mbps is guaranteed for lengths up to 40 m. Up to 500 m lengths of cable may be used if the data rate is reduced to 125 Kbps.

An example of a small CAN controller chip is the MCP2510 by Microchip. It is an 18-pin standalone CAN controller featuring an industry standard SPI serial interface. On-board features include interrupt capability, message masking and filtering, message prioritisation, and multi-purpose I/O pins. Multiple transmit/receive buffers significantly offload the microcontroller overhead required to handle CAN message traffic. Applications for the MCP2510 include device control, sensor monitoring, meter interfacing, automotive electronics, and instrument control.

## 5.9 Inter-IC Serial Buses

Working with embedded systems, one often comes across several serial buses used between ICs or modules within a system. These are typically de facto standards originally developed by IC manufacturers for specific functions and subsequently expanded to more general use. Even though they are only used for short interconnections, they are increasingly important to embedded systems designers as miniaturisation and simplicity in designs are emphasised. For example, using a serial bus between a flash memory device and the microcontroller not only reduces the complexity in circuit design, but also brings down the overall cost as serial devices are typically cheaper because of the smaller package and the PCB would be significantly smaller. The only drawback in using a serial bus is of course transmission speed reduction. However, in many laboratory scale embedded systems, the data rates of ~100 Kbps to ~10 Mbps available in serial buses are more than sufficient.

## 5.10 I2C (Inter Integrated Circuit Bus)

This is a bidirectional two-wire serial bus introduced by Philips Semiconductor in the 1980s for their television and other audiovisual products. Today it is widely used in the industry for embedded systems.

The two active lines are the SDA (serial data) and SCL (serial clock). A device has a unique address (7 or 10 bits) and may be receiver-only, or transmitter/receiver. Each device may be either a master or slave depending on whether it initiates a data transfer. Multiple masters are allowed in a bus. Over the years, data transfer speed has increased from 100 Kbps (standard) to 400 Kbps (fast) and 3.4 Mbps (high speed).

A bus master places the address of the destination (slave) on the bus. All devices on the bus listen and the one being addressed communicates with the master. If more than one master transmits, an arbitration scheme is used to decide the priority.

## 5.11 SPI (Serial Peripheral Interface)

This is a bidirectional full-duplex four-wire serial bus used originally by Motorola in their microcontrollers. The four wires are a clock, a data in, a data out and a select signal. The generic names of the SPI input/output (I/O) pins are:

- SS (slave select)
- SPSCK (SPI serial clock)
- MOSI (master out slave in)

- MISO (master in slave out)

In full duplex operation, the MISO pin of the master SPI module is connected to the MISO pin of the slave SPI module. The master SPI simultaneously receives data on its MISO pin and transmits data from its MOSI pin. Slave output data on the MISO pin is enabled only when the SPI is configured as a slave. To support a multiple-slave system, a logic 1 on the SS pin puts the MISO pin in a high-impedance state.

The MOSI pin of the master SPI module is connected to the MOSI pin of the slave SPI module. The master SPI simultaneously transmits data from its MOSI pin and receives data on its MISO pin.

The serial clock signal from master to slave synchronizes data transmission between the two. A byte of data is exchanged in 8 clock cycles.

The SS is used to select a slave when a device is configured in slave mode. When an SPI is configured as a master, the SS input can be used in conjunction with a flag to prevent multiple masters from driving MOSI and SPSCK, thus resolving contention.

## 5.12 SCI (Serial Communications Interface)

This is the Motorola implementation of the RS-232 asynchronous serial communications bus. It is a partial implementation, in that only the two I/O data lines are implemented, and they are at TTL voltage level instead of the standard bipolar signals as in the RS-232 specifications.

For short distance connections, such as within a piece of equipment, direct TTL levels are sufficient. If long transmission distance is necessary, or if the other end of the connection is a standard RS-232 port such as the PC COM port, voltage translation between TTL and RS-232 can be done.

Several manufacturers produce chips for this purpose. An example is the MAX232A by Maxim. It operates from a 5V power supply and generates typical voltages of +8 V and –8 V using charge pump voltage doubler and voltage inverter.

## 5.13 Microwire

This is a serial interface bus defined by National Semiconductor and used in many of their products including the COPS microprocessors. It is a three-wire (SO, SI and CK) interface very similar to the SPI. In fact many microprocessors with SPI interface may be used to connect to Microwire memories and peripheral devices.

## 5.14 1-Wire Bus

This is a serial bus defined by Dallas Semiconductor. 1-Wire devices lower system cost and simplify design with an interface protocol that supplies control, signalling, and power over a single-wire connection. A variety of identification, sensor, control, and memory functions are available in conventional IC packages, and stainless-steel-clad casing called iButtons.

A 1-Wire network consists of a master and one or more slave devices connected together through the 1-Wire interface. The master initiates and controls half duplex data transfer on the bus. It uses conventional TTL voltage levels of maximum 0.8 V for logical 0 and minimum 2.2 V for logical 1. The master has a weak resistive pull-up open drain output. A slave short circuits the data line to change the logical state to 0. For large networks, the weak pull-up is supplemented by a controlled strong pull-up.

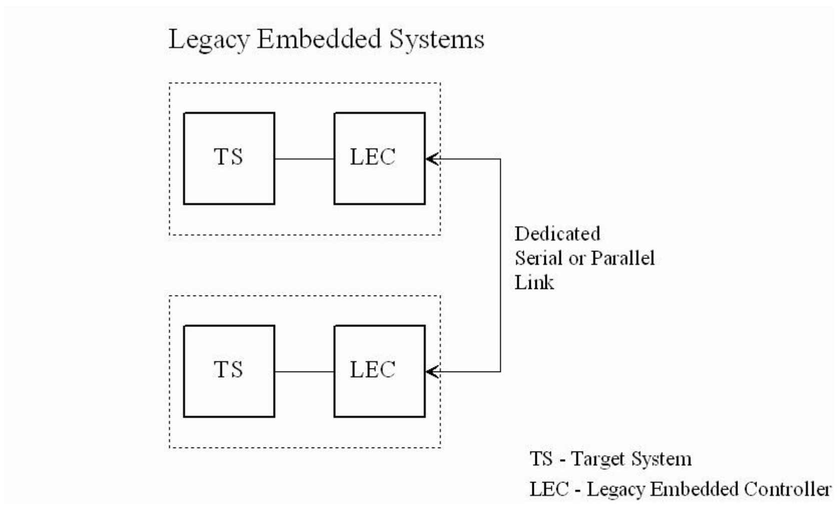
Every slave device has a unique 64-bit address, which consists of an 8-bit family code, a 48-bit serial number and an 8-bit CRC of the first seven bytes.

Slave devices must have their own timing circuits, synchronised by the falling slope of the signal on the bus. A slave typically obtains its power from the 1-Wire bus by means of an on-chip capacitor.

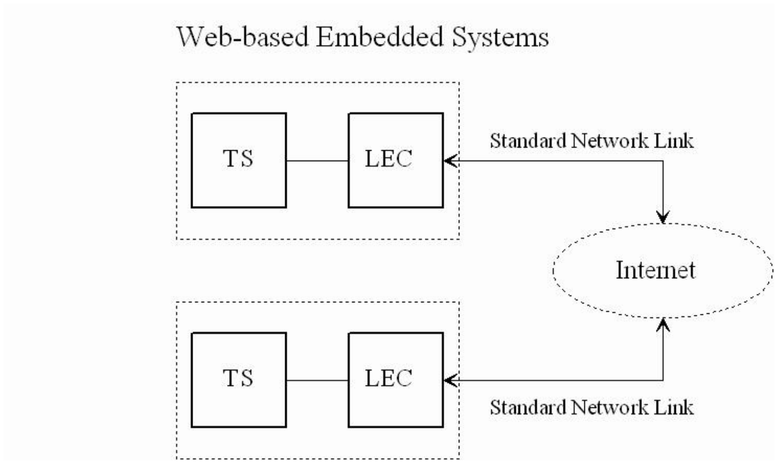
# 6 Web-based Embedded Systems

## 6.1 Introduction

Traditional networking of embedded systems is through the use of dedicated serial or parallel link as show below. A number of industrial bus standards are available. However, more often than not, straightforward non-standard data link layer protocols are often used in most laboratories for ease of implementation.


Legacy Embedded Systems
TS
LEC
TS
LEC
Dedicated Serial or Parallel Link
TS - Target System
LEC - Legacy Embedded Controller

The proliferation of the Ethernet and Internet brings another dimension to networking embedded systems. In a nutshell, the suite of Internet protocols and the cost reduction in embedded Ethernet hardware changes the way embedded systems are connected in a distributed environment. It is now cheaper and simpler to connect an embedded system, even a very simple and small one, to another piece of equipment via the Internet (or its variation the Intranet), than to link the two together using a traditional direct connection.



When the equipment is within the range of a LAN (typically ~ 100 m), the Ethernet is the choice for physical connection. To cover large distances, the Ethernet will typically have one of several ways to connect to the public telecommunications network. Thus the embedded system is connected to the WAN (wide area network). The links to the telecommunications network can be modems, ISDN, DSL, or leased lines. These links are not the job of the embedded systems designers. The job of the designers boils down to adding the Ethernet connection to the equipment, or a serial link to the modem if Ethernet is not used.

The suite of Internet protocols (UDP and TCP/IP and application protocols FTP, HTTP, SNMP etc.) is covered by other lectures in this workshop, so we shall not delve into this topic in any depth. However, the Ethernet interface and the serial link that connect the embedded system to the Internet shall be discussed here.

The last few years have seen the appearance of experimental, hobbyist and commercial real-time embedded systems on the Internet that can be accessed easily using standard browsers. However, one of the earlier implementations is the Cambridge coffee pot webcam that appeared in 1991 (predated world wide web) and was subsequently put on the www. (http://www.cl.cam.ac.uk/coffee/coffee.html) Now there are numerous webcams on the Internet. At the height of its fame, there were more than 2 million people viewing the plain coffee pot in the Trojan Room of the Computer Laboratory in Cambridge.

While it is fascinating and significant to be able to view video frames from any place in the world with an Internet connection, the basic principle is straightforward and simple. A server gets data (video frames in this case) ready in a form meaningful to the client (the web browser) and the client accesses it via the

Internet. It turns out that the server side effort is relatively simple (writing an HTML document, with SSI server side includes). The client (the browser) is a complex program but it is a standard application in all platforms. In fact, the earliest Cambridge coffee pot server and client implementation was done in one day, at that initial stage, for fun!

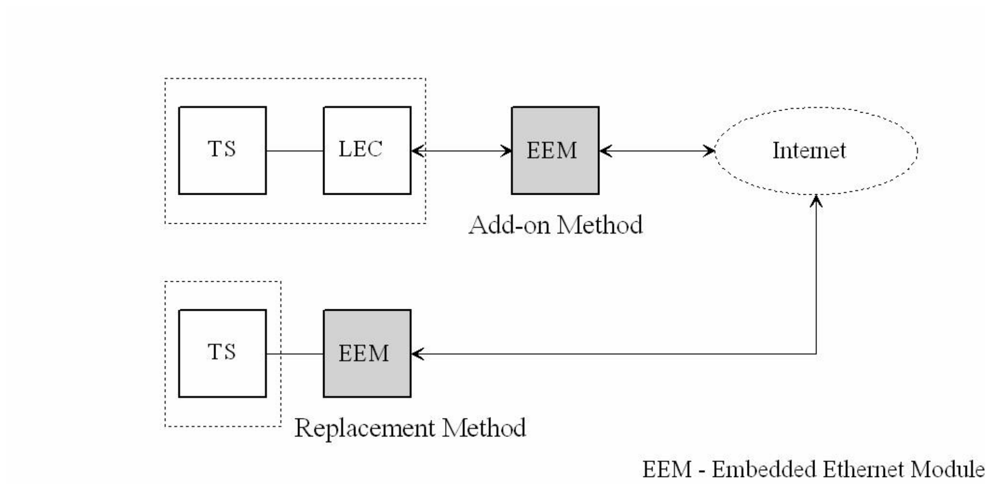
The more recent implementations of web-based applications that are of interest to the embedded community is in the implementation of small and lean embedded web servers. These are standalone servers that have their own IP (Internet address) and are connected to the Internet without using a PC. One of the tiniest is a match head sized web server using an 8-pin PIC microcontroller (PIC12C509A). (http://www-ccs.cs.umass.edu/~shri/iPic.html) It has a TCP/IP stack and a HTTP web-server. The TCP/IP stack is fitted into 256 bytes. This project proves the point that putting up a web-based embedded system, albeit a rather basic one, can be done with very little hardware, and software!

More realistic embedded web servers are built with microcontrollers with more memories than that of the PIC12C509A, which has a mere 1536 bytes of ROM and 41 bytes of RAM. Almost every microcontroller manufacturer now has application notes on how to implement a TCP/IP stack on their range of microcontrollers.

Vendors and suppliers of embedded web servers are mushrooming in the Internet because of the potential market they see. At the moment embedded Ethernet modules that have memories in the range of 256 Kbytes to 1 Mbytes, an Ethernet link, several serial links, and parallel I/Os are available in the price range of $50 ~ $100. For most laboratory applications where the volume is low, this is the most cost effective way to implement embedded web-based systems.

## 6.2 Migration to Web-based Systems

The embedded systems designer faces a dilemma – whether to build the complete application on the new embedded web-based modules or to retain the existing or legacy systems. As in the case of other technological innovations, there are several approaches to handling the migration to the new technology. In embedded systems at this time, there are two ways as shown below.



The first decision that has to be made is whether to use the new embedded web-based microcontroller module as:

(1) An *add-on* to the legacy system or
(2) A *replacement* of the legacy electronics.

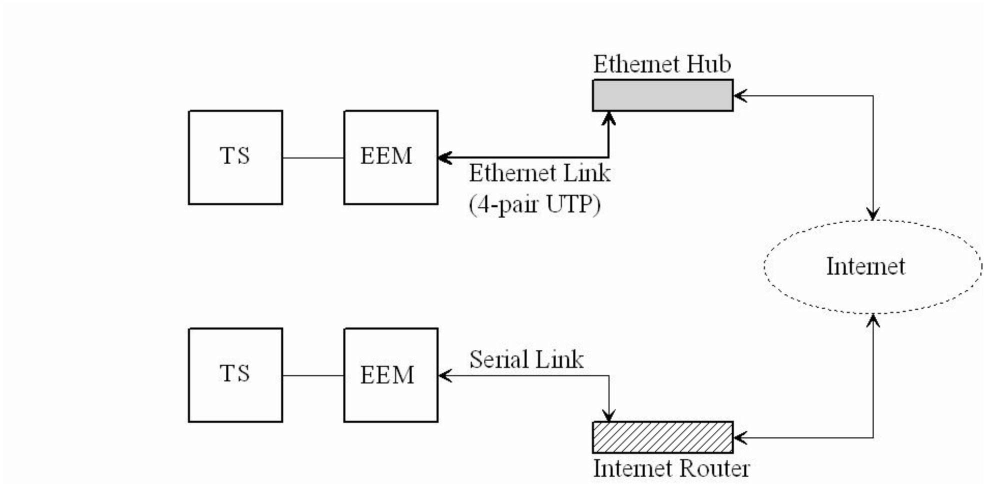
The add-on option has the advantage of little or no disruption to the existing system. In many cases where the existing systems have standard serial or parallel I/Os, no hardware or software modifications are needed. In others, appropriate modification to bring the existing system peripheral acceptable to the embedded Ethernet module has to be done. This is straightforward in many laboratory type systems. This is also a good option to adopt if the legacy system is a complex piece of equipment

The shortcoming of this add-on option is the lack in functional enhancement of the final system other than the Internet connection. Of course the Internet connection may be all that is needed, in which case it is obviously the simplest solution.

The replacement option is to replace part or all of the existing controller electronics with the embedded module. This is an attractive solution in some cases because new functions may be added to the existing system using the more powerful embedded microcontroller. Cost may be another factor. The new crops of embedded microcontroller modules are substantially cheaper than the older microprocessor-based systems. Thus if additional systems are needed, the old hardware may be expensive or obsolete. However, the effort and cost of rewriting or porting the software and firmware have to be weighed carefully against the gain achieved through hardware replacement.

## 6.3 Hardware Connection to the Internet

Irrespective of the way – add-on or replacement – the new embedded web-based module functions in the legacy system, there are two alternatives for connecting to the Internet world:



(1) Ethernet connection
(2) Serial connection

The Ethernet is now so widely deployed that there is hardly any laboratory or office that does not use it to network PCs or other equipment. Thus, for embedded systems, an *Ethernet capable* feature would simplify connection to the Internet. A 10Base-T or 100Base-T connection from an embedded system will be most convenient; the connection to the outside world then reduces to linking the system to a hub or switch with a simple CAT-5 UTP (unshielded twisted pair) cable.

Until recently, the rather complex CSMA/CD physical layer protocol (MAC) and the large buffer size (~ 1500 bytes) required for the Ethernet frame meant that implementation was costly. Interface to the Ethernet controller chips was typically designed for 16- or 32-bit buses which were rather uncommon in small and medium scaled embedded systems. Many small embedded systems thus cannot justify having an Ethernet interface. The situation has changed in the last two years. Both cost reduction and simplification of driving circuitry make it now possible to tug an Ethernet interface on an embedded system. Only two key components are needed – an Ethernet controller chip and a line driver transformer. An 8-bit microcontroller can now be used to interface to the controller chip. The cost of the Ethernet controller and the line driver is in the range of $10 ~ $20.

While the Ethernet interface is the ubiquitous link to the Internet, it is not the only way. An embedded system can be connected to the outside world via a serial link like the RS-232 COM port on the PC. There are a number of situations where this method of connection is preferred. First, it is cheaper to use a serial port, which is a standard item in most microcontrollers. This cost reduction refers to the embedded system side. It is noted that a matching serial port (in the form of Internet router) is needed on the network side. This may increase the overall system cost because an Internet router with serial ports has to be installed. Such Internet routers are not as common as the Ethernet hubs in most laboratories. Dispensing with the Ethernet controller chip reduces both the hardware and software complexity needed on the microcontroller host. In the tiny web server project mentioned earlier using a small PIC controller, there are simply not enough hardware resources on the microcontroller to drive an Ethernet controller.

Another reason for deploying a serial link is to connect to a modem for Internet access. This is a very common mode of connection. Most of us use a modem dial-up to access the Internet at home. Embedded systems to be used at locations without an Ethernet can use the serial mode in a similar manner.

In a very small implementation of a single embedded system connected to a PC acting as a host processor or controller, the serial link is also the choice. In this case, the PC host's COM port is simply connected to the serial port of the embedded system using a crossover cable (null modem).

## 6.4 Ethernet Connection

While there are several types of Ethernet connections and several high speed Ethernets (e.g. 10 Gbps) being introduced and standardized, the type that concerns the embedded systems designer is the plain 10Base-T, which is 10 Mbps, base-band, with a twisted pair. This is the most widely used method of accessing the Ethernet in most laboratories. In a typical network, switches or hubs are used to link the various Ethernet nodes. An embedded system with a 10BaseT port only needs an Ethernet cable (CAT-5, UTP with modular RJ45 plugs at both ends, up to 100 m) to connect to an available port on a hub.

In the event that you have to build your own Ethernet for your embedded systems network in your laboratory, what hardware components are involved? First you typically need a PC with an Ethernet connection. Many new PCs in the market have a built-in Ethernet port. For older PCs, a PCI Ethernet card (less than $20) can be used. Then you need an Ethernet hub which connects all nodes in your network. Hubs come in various forms. A small one with 4 to 8 ports

($20 ~ $50) would normally be sufficient. Then of course you need the cables with RJ45 terminations.

As in the case of serial link, it is possible to connect a host PC to an embedded system with just a crossover cable. No Ethernet hub is necessary in this case. This turns out to be a handy set-up for testing and debugging your embedded system. Before you deploy your embedded system on the Internet, you may want to test your system by just connecting it to a standalone host PC with a simple crossover cable.

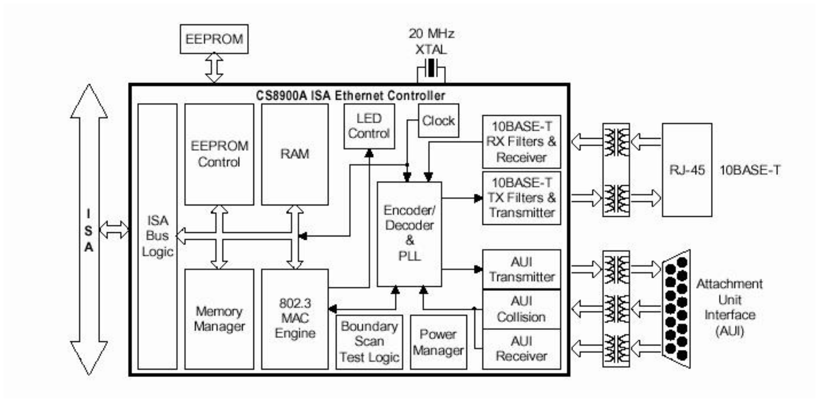
## 6.5 Ethernet Controller

Most PC users are familiar with the NIC (network interface card), which provides the interface to the Ethernet. Two widely used cards are the 3Com 3C509 and the Novell NE2000. These cards, typically connected to either an ISA or PCI bus, use Ethernet controller chips, which are rather sophisticated. The full functions of these Ethernet controller chips are often not required in embedded systems. Although the controller chips are rather complex, they nevertheless can be implemented in small 8-bit microcontroller systems with relative ease.

One such controller is the RTL8019AS produced by Realtek, which is the most widely used with 70% world market share in 1999. Both controllers are simple to use and relatively low cost. Packaged in a 100-pin PQFP (plastic quad flat pack) measuring 14 by 20 mm the RTL8019AS has the following main features:

- 16 Kbytes SRAM
- IEEE802.3 compliant
- Software compatible with NE2000
- PnP
- Full-duplex
- Power down modes
- Supports BROM
- Diagnostics LED outputs

Another similar controller is the CS8900A by Cirrus Logic. Originally designed as an ISA-bus Ethernet controller, it can be adapted for 8-bit microcontroller use. A complete Ethernet circuit can be designed on a PCB of about 10 cm$^2$. A block diagram of this chip is shown below:



The controller is accessed through either 8- or 16-bit ports. There are 8 registers which can be memory mapped into the usual address space. Each register is 16 bits. For embedded system applications, those registered are accessed directly by the microcontroller.

| Address Offset | I/O | Register |
|---|---|---|
| 0h | R/W | Receive/Transmit Data (Port 0) |
| 2h | R/W | Receive/Transmit Data (Port 1) |
| 4h | W | TxCMD (Transmit Command) |
| 6h | W | TxLength (Transmit Length) |
| 8h | R | Interrupt Status Queue |
| Ah | R/W | PacketPage Pointer |
| Ch | R/W | PacketPage Data (Port 0) |
| Eh | R/W | PacketPage Data (Port 1) |

The Ethernet frame transmission is carried out as follows:

(1) To transmit a frame, first write a transmit command (00C0h) to the TxCMD port and the length to the TxLength port. In the case of 8-bit connection, each frame is done in two steps, a low byte and then a high byte to the appropriate address. The BusTX register is then checked to see if the transmit buffer is available. This is done by setting the Packet-page Pointer with the correct value (in this case 0138h) and checking the PacketPage Data (Port 0) for the appropriate status, in this case bit 8 (Rdy4TxNow flag).

(2) If a transmit buffer is available (Rday4TxNow flag set), data are written, one byte at a time, to the Receive/Transmit Data (Port 0), two bytes in two consecutive locations of the port.

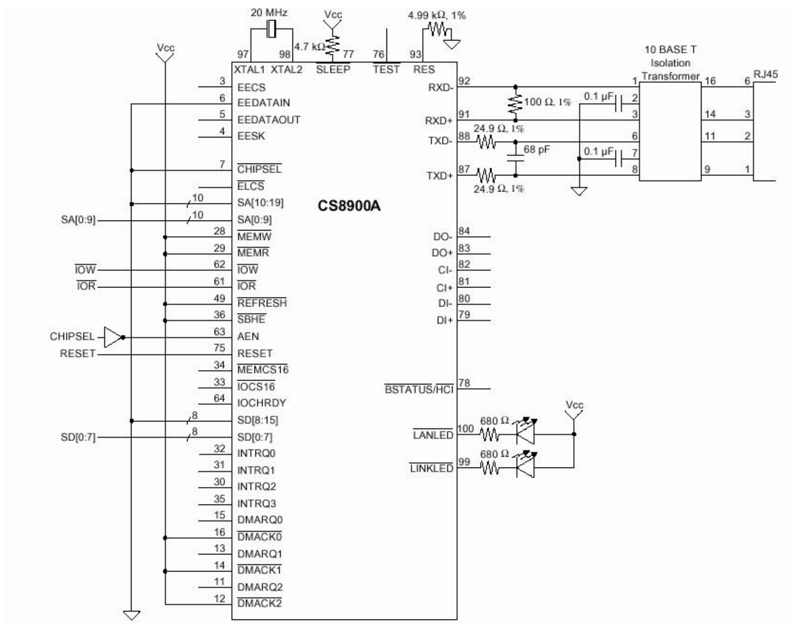
The steps for frame reception are:

(1) Poll Rx Event Register to determine if a frame is ready to be read. This is done by reading the RxStatus word from data port 0, high order byte first.

(2) The frame length is then read from data port 0, again high order byte first.

(3) The frame data is then read from data port 0, low order byte first. Repeat until the whole frame is read.

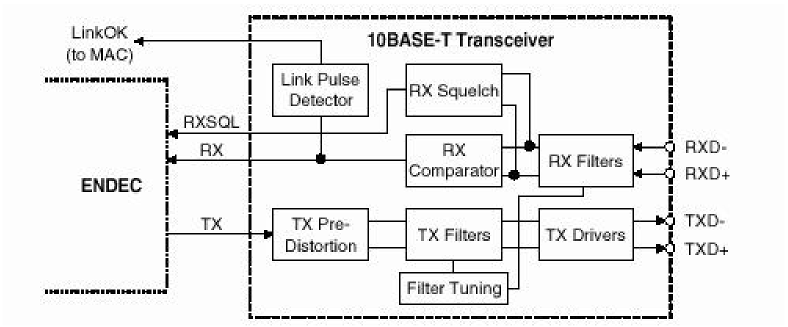
One can see the simplicity of interfacing the controller from the following reference design for an 8-bit microcontroller using the CS8900A.



Two LED outputs are usually provided by the Ethernet controller to monitor line status as shown in the diagram above. A LANLED is turned on (logical low) when the controller transmits or receives a frame, or when a collision is detected. It remains low until there has been no line activity for 6 ms.

A LINKLED is controlled by either the controller or the host. In the former, this LED is turned on whenever there are valid 10Base-T pulses. In the latter case, this LED is turned on whenever a HCB0 bit (a bit in a SelfCTL register) is set.

Ethernet controllers such as the CS8900A have a built-in 10Base-T transceiver including analogue and digital circuitry needed to connect to a simple isolation transformer. A block diagram of the transceiver is as follows:



Fifth-order Butterworth low-pass filters are used for the receiver and transmitter. The nominal 3 dB cutoff frequency of the filters is 16 MHz and the attenuation at 30 MHz is –27 dB.

In the 10Base-T transmitter, Manchester encoded data from the ENDEC (encoder decoder) passes through a predistortion circuit for wave shaping and pre-equalization. The signal is then filtered before being fed to differential drivers and finally brought out to the TxD+ and TxD- pins. Link pulses are sent in the absence of transmit packets. These are positive pulses, one bit time wide, generated every 16 ms.

In the receiver, a squelch circuit determines if the incoming filtered signal is valid (reaches the threshold). The receiver pair RxD+ and RxD- is monitored continuously. If a packet or link pulse is not received within a time limit (150 ms), transmission of packet is disabled. The received signals are also checked for polarity. In the case of polarity reversal, it is possible to set the controller to correct for the reversal.

## 6.6 On Simple Laboratory Measurements Using Embedded Processors

The really basic measurements in physics laboratories are simply time (t) and voltage (V).

It is obvious that the time of occurrence of an event must be recorded. All other physical parameters can be converted to voltages using transducers and appropriate electrical circuitries. A transducer may produce a change in electrical voltage directly when the physical parameter varies (e.g. thermocouples and piezoelectric transducers). However, it is more common to encounter a transducer that causes one or more of the basic electrical parameters (resistance R, capacitance C and inductance L) to change.

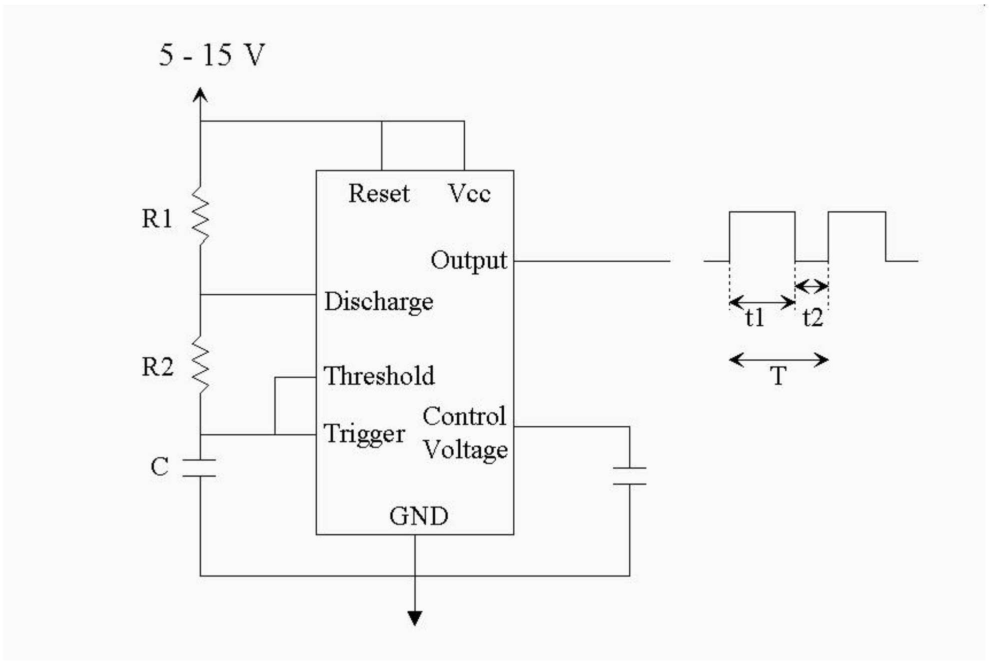
Embedded systems, perhaps with a few additional components, are extremely good at carrying out such measurements.

Event counting and time measurement used to be rather tedious and often expensive because of the dedicated circuitries needed. They are now easily achieved, with microsecond resolution, using common microcontrollers. Such time resolution is sufficiently high for a great variety of experiments in physics laboratories. Interestingly, real-time measurements at microsecond resolution can be achieved with very lost cost embedded systems such as PIC microcontrollers rather than with a high end PCs because of the heavy software overhead and the non-real-time nature of the OS in PCs.

Another nice feature of the small microcontrollers is the built-in analogue-to-digital (ADC) function. Often four or eight multiplexed ADC inputs are available, typically with 8-bit to 10-bit resolution. Acquisition is often in the microseconds or tens of microseconds using successive approximation technique. These translate to about 20K – 200K samples per second, a rather impressive performance for microcontroller costing about one dollar or one euro. There are many situations when such ADCs are adequate in laboratories and an example has been given earlier in a motion detector application.

In some situations, precision voltage measurements cannot be avoided. This was again a costly and complicated business. However, the situation has changed quite dramatically in the last few years. It is now possible to acquire a small ADC chip with up to 17 bits resolution (using dual slope conversion) for about \$3 (Microchip TC500A) and together with a precision voltage reference IC (MCP1525 at about \$0.5), one has an ADC subsystem capable of measuring physical parameters up to a precision of 1 part in 10^5, or 0.0008%. Some serious experiments can be done with such precision. One limitation of this low-cost high-precision ADC is of course the speed of conversion. It can typically perform four to seven conversions per second.

As mentioned earlier, most physical parameters can be converted to voltages by means of transducers and the job then boils down to measuring the electrical voltages with an appropriate ADC circuit. However, in practice there is an alternative and rather simple solution in place of the ADC, if the transducer is a resistive or capacitive type. A rather versatile timer IC, 555, first introduced by Signetics in 1971, has been used to measure resistance and capacitance with good precision at very low cost. Even though the design is 30 years old now, a large number of its derivatives are still being used widely for timing and measuring of R and C. The success of this IC is due to its simple, elegant and ingenious design that allows R and C to be measured accurately, independent of the IC components and power supply variations. Typically, the accuracy is dependent on only one or two external components over which the user has control. In the astable multivibrator mode the period of oscillation is given by the famous relation:



t1 = 0.693 (R1 + R2) C
t2 = 0.693 (R2) C

T = 0.693 (R1 + 2 R2) C

where R1, R2 and C are the external components of the circuit.

Since the period t is independent of the supply voltage and the internal IC parameters, rather high precision can be achieved if the external components are precise and stable. For example, to achieve an accuracy of 1 part in 10^6, one only has to set the period to about one second and measure it with a resolution of one microsecond. This is easily achieved with microcontrollers as we know. Thus a capacitive transducers that has a nominal capacitance of 1 uF requires only R1 and R2 to be in the range of M ohms and stable to within the accuracy required by the user.

By means of the 555 timer, we have transformed the problem of accurate measurements of physical parameters to time measurements using low cost microcontrollers such as PIC, TINI or RabbitCore. An example using such as circuit for temperature measurement is given below.

Being able to time physical events and read voltages is only doing part of the job of an instrumentation/control system. In most cases, some kinds of outputs must be carried out too. Once again, microcontrollers with its parallel ports are very good at producing digital outputs and sending out digital control signals at microsecond precision. In physics laboratories, very often, software controllable variable voltage sources are required. This calls for digital to analogue conversion (DAC). It is typically done outside a microcontroller. Fairly low cost DACs are available nowadays. For example, a 10-bit DAC, TC1321 from Microchip costing $1.5, can provide analogue voltage output up to about 100K samples per second with an accuracy of 1 part in 1000 or 0.1%. This again is sufficient for a rather large range of experiments in laboratories.

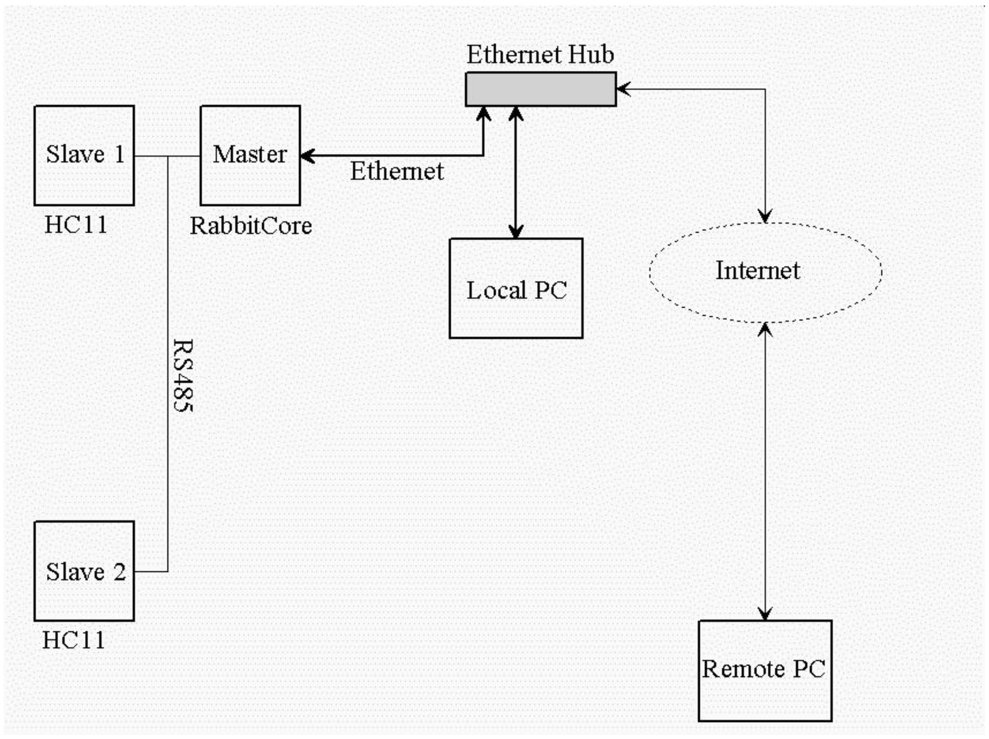
## 6.7 An Example of a Distributed Embedded System

A web-based distributed embedded system is given in this section to illustrate the technique of interfacing simple legacy equipment via the Internet. For simplicity, relatively small amount of hardware is involved. A RabbitCore is connected to two legacy devices, each an HC11-based board, in a master slave configuration where the RabbitCore is the master. To illustrate the technique of implementing a small distributed system, these two legacy devices are linked to the RabbitCore via an RS485 serial bus. Thus they may be as far apart as 4000 ft, simply connected via a pair of twisted wires.

Each HC-11 board has an input device and an output device. The input device is a 555 timer while the output device consists of three LEDs of different colours. The 555 timer can be used as (1) a thermometer, (2) a simple capacitance meter, (3) a simple ohmmeter and (4) pulse generator. A rotary switch is used to select one of these functions manually.

The RabbitCore communicates with the HC-11boards via a simple byte-oriented protocol as shown below. The master (RabbitCore) sends a command or message packet to the slaves whenever it wants to get temperature reading or control LEDs.

The RabbitCore is also connected to an Ethernet using a 10BaseT UTP connection. A web server is implemented so that other PCs can access the legacy devices using a standard web browser. If the RabbitCore is assigned an external IP, then Internet access is possible.



## Master Slave Protocol

The format of the packet sent from the RabbitCore (master) to the HC11 boards (slaves) is as follows.

| STX | DN | DT | DATA | ETX |
|---|---|---|---|---|

The first byte is Start of Transmission (STX), followed by a Device Number (DN) byte that specifies the device of interest. Thus DN=1 addresses the first device and DN=2 the second and so on. A simple broadcast address (e.g. 0 or FFh) may be assigned. The next field is Data Type to tell the addressed device the function needed, e.g. “T” for temperature measurement, “R” for resistant measurement, “C” for capacitance measurement and “F” for pulse width measurement, “L” for controlling the LEDs. The DATA field is parameter or data needed for a particular function, e.g. the LED pattern. Finally the packet is terminated with an End of Transmission.

The slave devices respond with the following data packet:

| STX | DT | DATA | ETX |
|---|---|---|---|

For simplicity error detection is done at the byte level using parity and framing error detection. The whole packet can be further protected with packet level check word such as checksum or CRC if necessary.

## Web Server Software

The main program in the web server in RabbitCore consists of an HTML page together with the necessary service-side include (SSI) and common gateway function interface (CGI) functions. The HTML page shows a graphical user interface depicting LEDs, switches (to toggle the LEDs), and temperature or other readings. SSIs are used to perform specific server-side function such as taking a temperature reading and displaying the value on the page. Examples:

```
<!--#exec cmd=”read_device0”-->
```

This SSI tag requests the server to execute a `read_device0` function that presumably will send a command to the slave device 0 to take a reading.

```
<!--#echo var=”reading0”-->
```

This SSI tag puts the reading0, which presumably is the returned reading from the slave device 0, into the HTML output string.

```
<A    HREF=”/toggle_green_led.cgi”>    <img    SRC=“but-
ton.gif”></A>
```

This tag shows how a button image hyperlink generates an HTTP requests to a CGI function `toggle_green_led.cgi.` This CGI function, upon execution, will send a toggle request to the slave device and also toggle the value of the green LED on the HTML page, by using different gif images.

## Client Side Software - Temperature Log

A simple program was implemented on a client PC to convert the system into a temperature logger. A simple web client is implemented using an HTTP GET request to the RabbitCore web server to get a temperature reading. The return value in the web page is parsed and logged down in a temperature log file. If this is executed periodically using, for example, the scheduler crond daemon in Linux, we then have a temperature logging system. The temperature logs can then be accessed in the raw form or presented to other web browser clients in the form of graphs.