Embedded Systems Memory

Each microcontroller has a specific addressing range. An addressing range is the number of addresses a microcontroller can access. The addressing scheme used to access to these spaces varies from processor to processor, but the underlying hardware is similar. Some processors/controllers have an on-chip memory while others may use an external (off-chip) memory depending upon the requirements.There are different types of memories available to be used in computers as well as embedded system and can be categorized based on their usage.

1. Program Storage Memory

Used primarily for program/code storage. Data at any memory location can be only read (ROM). Depending upon the fabrication, erasing and programming techniques they are classified as follows:

- Masked ROM (MROM)

Uses hardwired technology for storing data by masking and metallization at the time of production. The binary code for the program is taken as input by the MROM fabricator and accordingly the integrated circuit is created.The main advantage of MROM is low cost of production.

- Programmable ROM (PROM/OTP)

Not pre-programmed by the manufacturer and mainly has a nichrome or polysilicon wire matrix.Can be programmed by the end user only once by burning certain fuses using specialized PROM burners (One time Programmable).In the un-programmed state the data is entirely made up of 1's and blowing the desired fuses enables creating logic 0's.

- Erasable PROM (EPROM)

Programmable by the end user multiple number of times in case of program modification and reusability.Stores the bit information by charging a (Field Effect Transistor) FET. Erasing is performed by exposing the chip to a source of ultraviolet light (takes 20 to 30 minutes).

- Electrically Erasable PROM (EEPROM)

It is same as EPROM, but the erase operation is performed electrically.Any byte in EEPROM can be erased and rewritten as desired in a few milliseconds. Usually limited capacity.

- FLASH

Flash memory devices are high density, low cost, non-volatile, fast (to read, but not to write), and electrically reprogrammable. Organized as sectors or pages which can be selectively erased without effecting other sectors or pages.

2. Data Read-Write Memory/RAM

Data at any memory location can be read or written.Volatile memory i.e. retains the contents only as long as electricity is supplied. Data access to RAM is very fast as compared to ROM due to implementation technology.

Two main types.

- SRAM (Static RAM)

Stores a bit in the form of voltage by using flip-flops to implement a cell.
Each SRAM cell is made up of 6 transistors (four for the latch and two for controlling the access). Fast in operation but high cost.

- DRAM (Dynamic RAM)

Stores a bit in the form of charge by using a MOS transistor gate and a capacitor.
High density and low cost but relatively slower access. DRAM has short data retention lifetime due to charge leak and requires periodic refreshing. The DRAM controller periodically refreshes the data stored in the DRAM. By refreshing the data several times a second, the DRAM controller keeps the contents of memory alive as long as power is supplied.

- NVRAM

NVRAM is usually just a SRAM with battery backup. When power is turned on, the NVRAM operates just like any other SRAM but when power is off, the NVRAM draws enough electrical power from the battery to retain its content. NVRAMs with a lifespan of 10 years are fairly common in embedded systems(example DS1744 32KB from Maxim/Dallas.

Other Hardware Components

Input Output Subsystem

Most microcontrollers have several digital I/O ports. Usually a port consists of eight or fewer bits, and the bits in these ports can be outputs, inputs, or often bit programmable as either input or output bits. If an Input/Output port has programmable I/O, it will have an associated data direction register DDRA, DDRB, and so forth. The ports are usually named PORTA, PORTB, and so forth. DDRA is associated with PORTA. Each bit in DDRA has a corresponding bit in PORTA. If a bit in DDRA is set, the corresponding bit in PORTA is an output. The same is true for

PORTB, PORTC, PORTD, PORTE, and so forth if these ports exist on the part. Usually microcontrollers use an architecture called memory mapped I/O. Each I/O device, input and output registers/ports, its control registers, and status registers are mapped into memory locations. I/O transactions require no special computer instructions. It is merely necessary to know the memory locations of the pertinent registers and the uses of the register bits to be able to handle any I/O function. The system gets the inputs by the read operations at the port addresses.

The system has output ports through which it sends output bytes to the real world. An output may be connected to an LED (Light Emitting diode) or LCD (Liquid Crystal Display) panel. A control system sends the outputs to alarms, actuators, furnaces or boilers. A robot is sent output for its various motors. The system sends the output by a write operation to the port address. A port pin can be made into an output. When this occurs, this pin becomes a latched output. In other words, when this bit is set it will remain set until it is reset by the program, and vice versa. Just because a port pin is designated to be an output does not mean that its state cannot be read by the computer. When a port is read in, the state of all of the outputs as well as the state of the inputs will be shown in the result. Some I/O pins are multiplexed and serve multiple functions. A system connects to external physical devices and systems through a parallel or serial I/O communication interfaces. De-multiplexers and multiplexers facilitate communication of signals from multiple channels through a common path. A system often networks to the other devices and systems through an I/O bus for example I^2C, CAN, USB, ISA, PCI etc.

Timers and Counters:

The timer module, which is strictly speaking a counter module, is an important part of every microcontroller, and most controllers provide one or more timers with 8 and/or 16 bit resolution. Timers are used for a variety of tasks ranging from simple delays over measurement of periods to wave form generation. The most basic use of the timer is in its function as a counter, but timers generally also allow the user to timestamp external events, to trigger interrupts after a certain number of clock cycles, and even to generate pulse-width modulated signals for motor control.

A timer is a device that generates a signal pulse at specified time intervals. A time interval is a "real-time" measure of time, such as 3 milliseconds. These devices are extremely useful in systems in which a particular action, such as sampling an input signal or generating an output signal, must be performed every X time units. Internally, a simple timer may consist of a register, counter, and an extremely simple controller. The register holds a count value representing the number of clock cycles that equals the desired real-time value. The counter is initially loaded with the count value, and

then counts down on every clock cycle until 0 is reached, at which point an output signal is generated, the count value is reloaded, and the process repeats itself. A counter is nearly identical to a timer, except that instead of counting clock cycles (pulses on the clock signal), a counter counts pulses on some other input signal.

.

Interrupt Subsystem

Microcontrollers tend to be deployed in systems that have to react to events. Events signify state changes in the controlled system and generally require some sort of reaction by the microcontroller. Reactions range from simple responses like incrementing a counter whenever a work piece crosses a photoelectric barrier on the conveyor belt to time-critical measures like shutting down the system if someone reaches into the working area of a machine. Assuming that the controller can observe the event, that is, there is an input line that changes its state to indicate the event, there is still the question of how the controller should monitor the input line to ensure a proper and timely reaction. It is of course possible to simply poll the input signal, that is, to periodically check for state changes. However, this polling has its drawbacks: Not only does it unnecessarily waste processor time if the event only occurs infrequently, it is also hard to modify or extend. After all, a microcontroller generally has a lot more to do than just wait for a single event, so the event gets polled periodically in such a way that the rest of the program can be executed as well. On the other hand, the signal may have to be polled with a certain maximum period to avoid missing events, so the polling code may have to be called from several places in the main program. It is already time-consuming to establish from which positions in the code the signal should be polled in the first place, and these positions must be reconsidered whenever the main code changes. Hence, polling soon loses its attraction and the software designer starts looking for other ways to handle these infrequent events.
Fortunately, the microcontroller itself offers a convenient way in the form of interrupts. Here, the microcontroller polls the signal and interrupts the main program only if a state change is detected. As long as there is no state change, the main program simply executes without any concerns about the event. As soon as the event occurs, the microcontroller calls an interrupt service routine (ISR) which handles the event. The ISR must be provided by the application programmer. An interrupts-handling mechanism must exist in each system to handle interrupts from various processes in the system: for example, to transfer data from a keyboard or a printer.

The UART

A UART (Universal Asynchronous Receiver/Transmitter) receives serial data and stores it as parallel data (usually one byte), and takes parallel data and transmits it

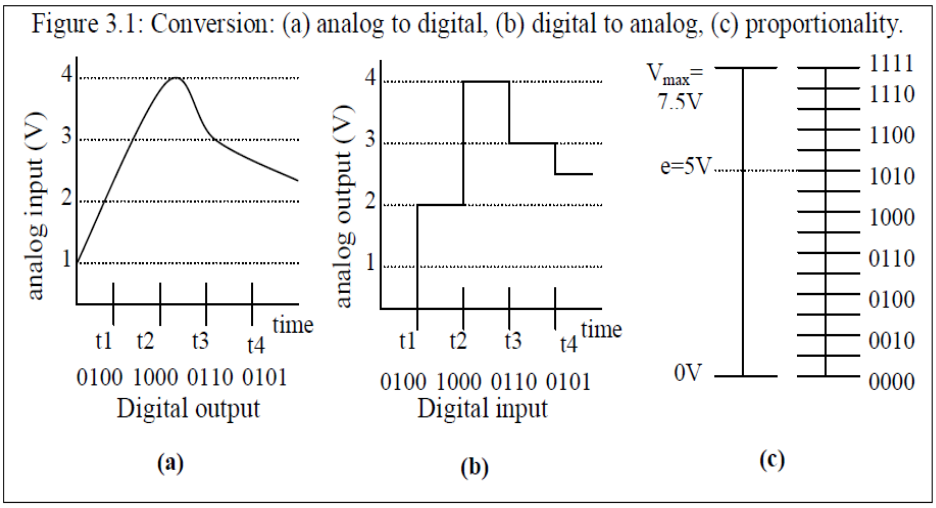
as serial data. Such serial communication is beneficial when we need to communicate bytes of data between devices separated by long distances, or when we simply have few available I/O pins.
We must be aware that we must set the transmission and reception rate, called the baud rate, which indicates the frequency that the signal changes. Common rates include 2400, 4800, 9600, and 19.2k. We must also be aware that an extra bit maybe added to each data word, called parity, to detect transmission errors -- the parity bit is set to high or low to indicate if the word has an even or odd number of bits. Internally, a simple UART may possess a baud-rate configuration register, and two independently operating processors, one for receiving and the other for transmitting. The transmitter may possess a register, often called a transmit buffer, that holds data to be sent. This register is a shift register, so the data can be transmitted one bit at a time by shifting at the appropriate rate. Likewise, the receiver receives data into a shift register, and then this data can be read in parallel. Note that in order to shift at the appropriate rate based on the configuration register, a UART requires a timer. To use a UART, we must configure its baud rate by writing to the configuration register, and then we must write data to the transmit register and/or read data from the received register.

PWM and Analog-Digital Conversion
Suppose a system needs to give an analog output of a control circuit for automation. The analog output may be to a power system for a D.C. motor or furnace. A Pulse Width Modulator (PWM) unit in the microcontroller operates as follows: Pulse width is made proportional to the analog-output needed. PWM inputs are from 00000000 to 11111111 for an 8-bit DAC operation. The PWM unit outputs to an external integrator and then provides the desired analog output.
PWM functionality could be implemented on a dedicated general-purpose processor, or integrated with another program's functionality, but the single-purpose processor approach has the benefits of efficiency and simplicity. An analog-to-digital converter (ADC, A/D or A2D) converts an analog signal to a digital signal, and a digital-to-analog converter (DAC, D/A or D2A) does the opposite. Such conversions are necessary because, while embedded systems deal with digital values, an embedded system's surroundings typically involve many analog signals.
We can compute the digital values from the analog values, and vice-versa, using the following ratio:

$$\frac{\mathrm{e}}{V_{max}} = \frac{d}{2^n - 1}$$

V*max* is the maximum voltage that the analog signal can assume, n is the number of bits available for the digital encoding, d is the present digital encoding, and e is the present analog voltage.

Internally, DACs possess simpler designs than ADCs. A DAC has *n* inputs for the digital encoding d, a Vmax analog input, and an analog output *e*. A fairly straight forward circuit (involving resistors and an op-amp) can be used to convert *d* to *e*.



a) Analog to Digital conversion (b) Digital to Analog conversion (c) Conversion range and levels.

ADCs, on the other hand, require designs that are more complex, for the following reason. Given a *Vmax* analog input and an analog input *e*, how does the converter know what binary value to assign in order to satisfy the above ratio? Unlike DACs, there is no simple analog circuit to compute d from e. Instead, an ADC may itself contain a DAC also connected to *Vmax*. The ADC "guesses" an encoding d, and then evaluates its guess by inputting d into the DAC, and comparing the generated analog output e' with the original analog input e (using an analog comparator). If the two sufficiently match, then the ADC has found a proper encoding. So now the question remains: how do we guess the correct encoding? This problem is analogous to the common computer-programming problem of finding an item in a list. One approach is sequential search, or "counting-up" in analog digital terminology. In this approach, we start with an encoding of 0, then 1, then 2, etc., until we find a match. Unfortunately, while simple, this approach in the worst case (for high voltage values) requires 2^n comparisons, so it may be quite slow.
A faster solution uses what programmers call binary search, or "successive approximation" in analog-digital terminology. We start with an encoding

corresponding half of the maximum. We then compare the resulting analog value with the original; if the resulting value is greater (less) than the original, we set the new encoding to half way between this one and the maximum (minimum). We continue this process, dividing the possible encoding range in half at each step, until the compared voltages are equal. This technique requires at most n comparisons. However, it requires a more complex converter.

Sensors and Actuators

A sensor is a transducer device that converts energy from one form to another for any measurement or control purpose for example a microphone converts sound energy into electrical signals. Actuator is a form of a transducer device (mechanical or electrical) which converts electrical signals to corresponding physical action for example a motor converts electrical energy into motion.

The commonly used sensors and actuators in embedded systems:

Switch/Button

The button is one of the simplest input elements. It consists of two contacts which are connected if the button is pressed. So if one of the contacts is connected to for example GND and the other is connected to the microcontroller input and to VCC through a pull-up resistor (either internally by the microcontroller, or with an external resistor), then the controller will read HIGH as long as the button is not pressed, and will read LOW while the button is pressed. However, a switch remains in the position it is put. So if the switch is put into its ON position (which connects the contacts), then it will stay on until it is moved into the OFF position. Push buttons can be connected to a microcontroller pin using a pull up or a pull down resistor.

Matrix Keypad

A keypad consists of a set of buttons that may be pressed to provide input to an embedded system. Again, keypads are extremely common in embedded systems, since such systems may lack the keyboard that comes standard with desktop systems. A simple keypad has buttons arranged in an N-column by M-row grid. Such an arrangement saves connections, so instead of n^2 pins for n^2 buttons, only 2n pins are required. The device has N outputs, each output corresponding to a column, and another M outputs, each output corresponding to a row. When we press a button, one column output and one row output go high, uniquely identifying the pressed button. To read such a keypad from software, we must scan the column and row outputs.

Light Emitting Diode (LED)

The LED (light emitting diode) is the most basic output element. Its form and color vary widely to accommodate a wide variety of applications. The color of a LED is determined by the chemicals used for it. Common colors are red and green, but yellow, orange, blue and white LEDs are also readily available, as well as LEDs emitting light in the infrared or ultraviolet bands.

Liquid Crystal Display (LCD)

An LCD (Liquid crystal display) is a low-cost, low-power device capable of displaying text and images. LCDs are extremely common in embedded systems, since such systems often do not have video monitors standard for desktop systems. LCDs can be found in numerous common devices like watches, fax and copy machines, and calculators.

Switching Loads

Although displaying status information is an important task, the real interest for employing microcontrollers in embedded systems lies in monitoring and controlling the environment. Monitoring is done with sensors, the subsequent control actions are executed through actuators. In order to be able to influence its environment, which may work on completely different power levels and require high amounts of current, the microcontroller needs some means to switch these loads. This can be done by incorporating the following:

- *Transistor*
- *Relay*
- *Optocoupler*

Motors

Electric motors consist of a pivotable core, the rotor, and a static casing, the stator. Depending on the type of the motor, the rotor is either a set of coils which must be excited externally, or a permanent magnet. The stator again is either a permanent magnet or consists of at least two and possibly more pairs of coils (the coils in each pair are situated opposite of each other) which are magnetized to provide the magnetic field necessary to turn the rotor. There are usually two types of motors used in embedded systems i.e. DC motors and Stepper motors.

Clock Oscillator and Reset Circuit

After the power supply, the clock is the next important unit of a system. A processor needs a clock oscillator circuit. The clock controls the various clocking requirements of the CPU, of the system timers and the CPU machine cycles. The machine cycles are for (i) fetching the codes and data from memory and then decoding and executing at the processor, and(ii) transferring the results to memory. The clock controls the time for executing an instruction. The clock circuit uses either a crystal (external to the processor) or a ceramic resonator (internally associated with the processor) or an external oscillator IC attached to the processor. The external IC-based clock oscillator

has a significantly higher power dissipation compared to the internal processor-resonator. However, it provides a higher driving capability, which might be needed when the various circuits of embedded system are concurrently driven. For example, a multiprocessor system needs the clock circuit, which should give a high driving capability and enables control of all the processors concurrently.

Reset means that the processor starts the processing of instructions from a starting address. That address is one that is set by default in the processor program counter (or instruction pointer and code segment registers in x86 processors) on a power-up. From that address in memory, the fetching of program-instructions starts following the reset of the processor. The processor circuit keeps the reset pin active and then deactivates to let the program proceed from a default beginning address.