Memory

The memory of the Atmel AVR processors is a Modified Harvard architecture, in which the program and data memory are on separate buses to allow faster access and increased capacity. The AVR uses internal memory for data and program storage, and so does not require any external memory.

The four types of memories in a Atmel AVR are:

- Data memory: registers, I/O registers, and SRAM
- Program flash memory
- EEPROM
- Fuse bits

All these memories are on the same chip as the CPU core. Each kind of memory is separated from each other, in different locations on the chip. Address 0 in data memory is distinct from address 0 in program flash and address 0 in EEPROM.

Program Memory

All AVR microcontrollers have some amount of 16 bit wide non-volatile flash memory for program storage, from 1 KB up to 256 KB (or, 512-128K typical program words). The program memory holds the executable program opcodes and static data tables. Program memory is linearly addressed, and so mechanisms like page banking or segment registers are not required to call any function, regardless of its location in program memory.

AVRs cannot use external program memory; the flash memory on the chip is the only program memory available to the AVR core.

The flash program memory can be reprogrammed using a programming tool, the most popular being those that program the chip *in situ* and are called in-system programmers (ISP). Atmel AVRs can also be reprogrammed with a high-voltage parallel or serial programmer, and via JTAG (again, *in situ*) on certain chips. The flash memory in an AVR can be re-programmed at least 10,000 times.

Many of the newer AVRs (MegaAVR series) have the capability to self-program the flash memory. This functionality is used mainly by bootloaders.

Data Memory

Data Memory includes the registers, the I/O registers, and internal SRAM.

The AVR has thirty-two general purpose eight-bit registers (R0 to R31), six of which can be used in pairs as sixteen-bit pointers (X, Y, and Z).

All AVR microcontrollers have some amount of RAM, from 32 bytes up to several KB. This memory is byte addressable. The register file (both general and special purpose) is mapped into the first

addresses and thus accessible also as RAM. Some of the tiniest AVR microcontrollers have only the register file as their RAM.

The data address space consists of the register file, I/O registers, and SRAM. The working registers are mapped in as the first thirty-two memory spaces (0000_{16}-$001F_{16}$) followed by the reserved space for up to 64 I/O registers (0020_{16}-$005F_{16}$). The actual usable SRAM starts after both these sections (address 0060_{16}). (Note that the I/O register space may be larger on some more extensive devices, in which case the beginning address of SRAM will be higher.) Even though there are separate addressing schemes and optimized opcodes for register file and I/O register access, they can still be addressed and manipulated as if they were SRAM.

The I/O registers (and the program counter) are reset to their default starting values when a reset occurs. The registers and the rest of SRAM have initial random values, so typically one of the first things a program does is clear them to all zeros or load them with some other initial value.

The registers, I/O registers, and SRAM never wear out, no matter how many times they are written.

External Data Memory

Some of the higher pin-count AVR microcontrollers allow for external expansion of the data space, addressable up to 64 KB. When enabled, external SRAM is overlaid by internal SRAM; an access to address 0000_{16} in the data space will always resolve to on-chip memory. Depending on the amount of on-chip SRAM present in the particular AVR, anywhere from 512 bytes to several KB of external RAM will not be accessible. This usually does not cause a problem.

The support circuitry required is described in the datasheet for any device that supports external data memory, such as the Mega 162, in the "External Memory Interface" section. The support circuitry is minimal, consisting of a '573 or similar latch, and potentially some chip select logic. The SRAM chip select may be tied to a logic level that permanently enables the chip, or it may be driven by a pin from the AVR. For an SRAM of 32 KB or less, one option is to use a higher-order address line to drive the chip select line to the SRAM.

EEPROM Storage

Almost all AVR microcontrollers have internal EEPROM memory for non-volatile data storage. Only the Tiny11 and Tiny28 have no EEPROM.

EEPROM memory is not directly mapped in either the program or data space, but is instead accessed indirectly as a peripheral, using I/O registers. Many compilers available for the AVR hide some or all of the details of accessing EEPROM. IAR's C compiler for the AVR recognizes the compiler-specific keyword `__eeprom` on a variable declaration. Thereafter, a person writes code to read and write that variable with the same standard C syntax as normal variables (in RAM), but the compiler generates code to access the EEPROM instead of regular data memory.

Atmel's datasheets indicate that the EEPROM can be re-written a minimum of 100,000 times. An application must implement a wear-leveling scheme if it writes to the EEPROM so frequently that it will reach the write limit before it reaches the expected lifetime of the device. AVRs ship from the factory with the EEPROM erased, i.e. the value in each byte of EEPROM is FF_{16}.

Many of the AVRs have errata about writing to EEPROM address 0 under certain power conditions (usually during brownout), and so Atmel recommends that programs not use that address in the EEPROM.

Fuse Settings

A fuse is an EEPROM bit that controls low level features and pin assignments. Fuses are not accessible by the program; they can only be changed by a chip programmer. Fuses control features which must be set before the chip can come out of reset and begin executing code.

The most frequently modified fuses include:

1. Oscillator/crystal characteristics, including drive strength and start-up time.
2. JTAG pins used for JTAG or GPIO
3. RESET pin used as a reset input, debugWire, or GPIO
4. Brown Out Detect (BOD) enable and BOD voltage trigger points

There is also a fuse to enable serial in-system programming, which is set by default. If it is set incorrectly, the only way to program the chip is by using a high-voltage programmer, such as the STK-500, AVR Dragon, or third-party programmer. A developer is therefore cautioned to be careful when manipulating fuses.

Reset

The AVR's RESET pin is an active-low input that forces a reset of the processor and its integrated peripherals. The line can be driven by an external power-on reset generator, a voltage supervisor (which asserts RESET when the power supply voltage drops below a predefined threshold), or another component in a larger system. For example, if the AVR is managing a few sensors and servos as part of a large integrated system, another controller might observe some condition that justifies resetting the AVR; it could do so by asserting the AVR's RESET line.

AVRs also include a watchdog timer, which can reset the processor when it times out. The watchdog timer must be reset periodically to prevent it from timing out. Failure to reset the watchdog timer is usually an indication that the program code has failed (locked up, entered an infinite loop, or otherwise gone astray), and the processor should be reset. On some AVRs the watchdog can be programmed to issue an interrupt instead of resetting the processor. This functionality can be used to wake up the AVR from a sleep mode.

The RESET pin is used for in-system serial programming, as a GPIO, or for debugWIRE™ low pin count debugging, depending on the chip and the programming of the fuse bits. If the reset functionality of that pin is disabled, it cannot be recovered by in-system serial programming, and another method such as high-voltage programming must be used.

Interrupts

AVRs support multiple interrupt sources, both internal and external. An interrupt could be from an internal peripheral reaching a certain state (i.e. character received on UART), or from an external event like a certain level on a pin. Each interrupt source causes a jump to a specific location in memory. That location is expected to contain either a RETI (Return from Interrupt) instruction to essentially ignore the interrupt, or a jump to the actual interrupt handler.

Most AVRs have at least one dedicated external interrupt pin (INT0). Older AVRs can trigger an interrupt on a high or low level, or on a falling edge. Newer AVRs add more options, such as triggering on the rising edge or either edge. Additionally, many of the newer AVRs implement pin-change interrupts for all pins in groups of eight, eliminating the need for polling the pins. The pin-change interrupt handler must examine the state of the pins that are associated with that interrupt vector, and determine what action to take.

Due to button bounce issues, it is considered poor design to connect a push button or other user input directly to an interrupt pin; some debouncing or other signal conditioning must be interposed so that the signal from the button does not violate the setup and hold times required on the interrupt pins.

General Purpose I/O Ports

General Purpose I/O, or GPIO, pins are the digital I/O for the AVR family. These pins are true push-pull outputs. The AVR can drive a high or low level, or configure the pin as an input with or without a pull-up. GPIOs are grouped into "ports" of up to 8 pins, though some AVRs do not have enough pins to provide all 8 pins in a particular port, e.g. the Mega48/88/168 does not have a PortC7 pin. Control registers are provided for setting the data direction, output value (or pull-up enabled), and for reading the value on the pin itself. An individual pin can be accessed using bitwise manipulation instructions.

Each port has 3 control registers associated with it, DDRx, PORTx, and PINx. Each bit in those registers controls one GPIO pin, i.e. bit 0 in DDRA controls the data direction for PortA0 (often abbreviated PA0), and bit 0 in PORTA will control the data (or pullup) for PA0.

The DDR (Data Direction Register) controls whether the pin is an input or an output. When the pin is configured as an output, the corresponding bit in the PORT register will control the drive level to the pin, high or low. When the pin is configured as an input, the bit in the PORT register controls whether a pull-up is enabled or disabled on that pin. The PIN (Port Input) register was read-only on

earlier AVRs, and was used to read the value on the port pin, regardless of the data direction. Newer AVRs allow a write to the PIN register to toggle the corresponding PORT bit, which saves a few processor cycles when bit-banging an interface.

Timer/Counters

All AVRs have at least one 8-bit timer/counter. For brevity, a timer/counter is usually referred to as simply a timer.

Some of the Tiny series have only one 8-bit timer. At the high end of the Mega series, there are chips with as many as six timers (two 8-bit and four 16-bit).

A timer can be clocked directly by the system clock, by a divided-down system clock, or by an external input (rising or falling edge). Some AVRs also include an option to use an external crystal, asynchronous to the system clock, which can be used for maintaining a real-time clock with a 32.768 kHz crystal.

In normal mode, the timer counts up to the top FF_8 (or $FFFF_{16}$), roll over to zero, and set an overflow bit, which may cause an interrupt if enabled. Within the interrupt routine, the counter can be loaded with the desired value in addition to any other processing required.

The Clear Timer on Compare (CTC) mode allows for the timer to be cleared when it matches a value in the compare register, before the timer reaches the top and overflows. Clearing the timer prior to overflow manipulates the timer resolution, allowing for greater control of the output frequency of a compare match. It can also simplify the counting of an external event.

The value of a timer can be read back at any time, even while it is running. (There is a specific sequence documented in the datasheets to read back a 16-bit timer so that a consistent result is returned, since the AVR can only move 8 bits at a time.) A timer can be halted temporarily by changing its clock input to "no clock source," then resumed by re-selecting the previous clock input.

PWM (Pulse Width Modulation)

Many of the AVRs include a compare register for at least one of the timers. The compare register can be used to trigger an interrupt and/or toggle an output pin (i.e. OC1A for Timer 1) when the timer value matches the value in the compare register. This may be done separately from the overflow interrupt, enabling the use of pulse-width modulation (PWM).

Some AVRs also include options for phase-correct PWM, or phase- and frequency-correct PWM. Phase correction is required by many motors.

The ATtiny26 is unique in its inclusion of a 64 MHz high-speed PWM mode. The 64 MHz clock is generated from a PLL, and is independent of, and asynchronous to, the processor clock.

Some AVRs also include complementary outputs suitable for controlling some motors. A dead-time generator (DTG) inserts a delay between one signal falling and the other signal rising so that both signals are never high at the same time. The high-end AT90PWM series allows the dead time to be programmed as a number of system clock cycles, while other AVRs with this feature simply use 1 clock cycle for the dead time.

Output Compare Modulator

An Output Compare Modulator (OCM), which allows generating a signal that is modulated with a carrier frequency. OCM requires two timers, one for the carrier frequency, and the second for the signal to be modulated. OCM is available on some of the Mega series.

Serial Communication

AVR microcontrollers are in general capable of supporting a plethora of serial communication protocols and serial bus standards. The exact types of serial communication support varies between the different members of the AVR microcontroller family.

On top of support in hardware there is also often the option to implement a particular serial communication mechanism entirely in software. Typically this is used in case a particular AVR controller does not support some serial communication mechanism in hardware, the particular hardware is already in use (e.g. when two RS-232 interfaces are needed, but only one is supported in hardware), or the chip's hardware can't be used, because it shares pins with other chip functions, and such a function is already in used for the particular hardware. The latter often happens with the low-pincount AVRs in DIP packages.

Finally, there is also the possibility to use additional logic to implement a serial communication function. For example, most AVRs don't support the USB bus (some later ones do so, however). When using an AVR which doesn't support USB directly, a circuit designer can add USB functionality with a fixed-function chip such as the FTDI232 USB to RS-232 converter chip, or a general-purpose USB interface such as the PDIUSB11. Adding additional electronics is in fact necessary for some supported communication protocols, e.g. standard-compliant RS-232 communication requires adding voltage level converters like the MAX232.

The number of serial communication possibilities supported by a particular AVR can be confusing at times, in particular if the pins are shared with other chip functions. An intensive study of the particular AVR's datasheet is highly recommended. The serial communication features most commonly to be found on AVRs are discussed in the following sections.

Universal Synchronous Asynchronous Receiver Transmitter (USART)

Recent AVRs typically come with a Universal Synchronous Asynchronous Receiver Transmitter (USART) built-in. A USART is a programmable piece of hardware which is capable of generating and decoding various serial communication protocols. USART is an acronym from the following words:

Universal

Can be used in a lot of different serial communication scenarios

Synchronous

Can be used for synchronous serial communication (sender and receiver are synchronised by a particular clock signal)

Asynchronous

Can be used for asynchronous serial communication (sender and receiver are not explicitly synchronised via a clock signal, but synchronise on the data signal).

Receiver

The hardware in the AVR can receive serial data

Transmitter

The hardware can send serial data

Earlier AVRs had a UART that did not support synchronous serial communication, hence the absence of the "S" in the acronym.

USARTs or UARTs work with logic voltage levels while e.g. the RS-232 protocol requires much different voltage levels than the 5 V or 3.3 V supplies found on AVR circuits. The conversion from and to such voltage levels is performed by an additional chip which is commonly called a line driver or line interface.

With the right line interface an AVR's USART can, for example, be used to communicate with RS-232, RS-485, MIDI, LIN bus, or CANbus devices, to name some of the popular protocols.

See Robotics: Computer Control: The Interface: Networks for more details.

RS-232 Signalling[edit]

The RS-232 specification calls for a negative voltage to represent a "1" bit, and a positive voltage to represent a "0" bit. The spec allows for levels from +3 to +15 V, and -3 to -15 V, but +/-12 V is commonly seen. The AVR does not have the ability to drive a negative output voltage on any GPIO pin, and so a level converter, such as the MAX232, is used to talk to PCs and strict RS-232 devices. See Serial Programming:RS-232 Connections for more detail on RS-232 wiring.

RS-232 has a relatively short maximum cable length. For longer cabling distances, consider using RS-485 signaling on your USART.

Two Wire Interface

TWI is a variant of Phillips' I²C bus interface. I²C consists of two wires, known as SDA (serial data) and SCL (serial clock), which use open-drain drivers and therefore require pull-ups to a logic-1 state. I²C uses a common ground, so all devices on the bus should be at the same ground potential to avoid ground loops. TWI uses 7 bit addressing, which allows for multiple devices to connect to the bus.

Many TWI devices have at least the top four bits of the address hard-coded, and the remaining bits configurable by some means such as connecting dedicated address pins to power or ground; this often allows for only 2-8 model *X* devices on the bus. The AVR's TWI hardware can act as Master or Slave, and can meet the 400 kbit/s spec.

Serial Peripheral Interface (SPI)

SPI, the Serial Peripheral Interface Bus, is a master-slave synchronous serial protocol. This means that there is a clock line which determines where the pulses are to be sampled, and that one of the parties is always in charge of initiating communication. It uses at least three lines, which are called:

MISO

Master In Slave Out.

MOSI

Master Out Slave In.

SCK

Serial Clock.

Conceptually, SPI is a bidirectional shift register; as bits are shifted out on either MISO or MOSI, bits are shifted in on the other line. The master always controls the clock.

An SPI slave has a Slave Select (SS) signal, which signals to the slave that it should respond to messages from the master. SS is almost always active-low. If there is only one master and one slave, the slave's SS line could be tied low, and the master would not need to drive it. If there are two or more slaves, then the master must use a separate slave select signal to each slave. The downside of this approach is that the master can only address as many slaves as it has extra outputs (without the use of a separate decoder).

Hardware Implementation The larger AVR microcontrollers have built-in SPI transceivers (from the ATmega8 upwards). The serial clock is derived from the processor clock, with several divisors available. The data length is always 8 bits. The clock polarity and phase may be configured, leading to four possible combinations of when the data is clocked in and out of the chip. This interface is very popular, and is widely available on a variety of other processors and peripherals.

The pins used for the SPI bus are also used as a way of programming the chip via ISP (In System Programming)(Except on the mega128).

Universal Serial Interface Some AVRs, particularly in the Tiny family, provide a Universal Serial Interface (USI) instead of an SPI. The USI is capable of operating as an SPI, but also as an I2C controller, and with a little extra effort, a USART. The bit length of the transfer is configurable, as is the clock driver. The clock can be driven by software, by the timer 0 overflow, or by an external source.

Software Implementation SPI can be implemented using bit-banging of the I/O lines. An efficient implementation of a slave can be done by connecting SCLK to an external interrupt source.

The datasheet for a particular AVR provides a block diagram of the SPI or USI controller on that chip.

Protocol Issues

SPI, RS-232, I^2C, and other serial interfaces only define the method by which bits and bytes are transmitted; they correspond to layer 1 in the OSI model, the physical layer. The bytes could be anything: temperature readings (in Celsius or Fahrenheit, depending on your sensor), readings from a pressure sensor, control signals to turn off a pump, or the bytes of a JPEG image. Some of this meaning may be assigned by the use of a serial communications protocol.

A serial protocol must handle a wide variety of usage conditions, as well as provide for recovering from failures. For example, if two sensors are connected to a single microcontroller (such as inside and outside temperature), the protocol provides a way for the receiver on the other end of the serial line to discern which reading belongs to which sensor. If a cable is unplugged during transmission, or a byte is lost due to line noise, the protocol can provide a way to re-synchronize the transmitter and the receiver.

The Serial Programming wikibook contains more discussion of serial protocols.

Analog Interfaces

Analog to Digital

Analog to digital conversion uses digital number to represent the proportion of the analog signal sampled. For example, by applying a 3 V to the input of an ADC with a full-scale range of 5 V, will result as a digital output of 60% of the full range of the digital output. The digital number can be represented in 8 or 10 bits by the ADC. An 8 bit converter will provide output from 0 to $2^8 - 1$, or 255. 10 bits will provide output from 0 to $2^{10} - 1 = 1023$.

10 bit sample: $\frac{3V}{5V} = 0.6 \times 1023 = 614$ in ADCH:ADCL or

8 bit sample: $\frac{3V}{5V} = 0.6 \times 255 = 153$ in ADCL

Many AVRs include an ADC, specifically a successive-approximation ADC. The ADC reference voltage (5 V in the example above) can be an external voltage, an internal fixed 1.1 V reference.

AVRs with an ADC have several analog inputs which are connected to the ADC via an analog multiplexer. Only one analog input can be converted at any given time. The ADC controller provides a method for sequentially converting the inputs, so that an AVR can easily cycle through multiple sources thousands of times a second. AVRs can run ADC conversions continuously in the background, or use a special "ADC sleep" mode to halt the processor while a conversion is taking place, to minimize voltage disturbances from the rest of the MCU.

Analog Comparator Peripheral

Nearly all AVR microcontrollers feature an Analog Comparator which can be used to implement an ADC on those AVRs which do not have an ADC, or if all of the ADC inputs are already in use. Atmel provides sample code and documentation for using the comparator as a low-speed ADC. The signal to be measured is connected to the inverted input, and a reference signal is connected to the non-inverting input. The AVR generates an interrupt when the signal falls below or rises above the reference value.

A common use for the analog comparator is sensing battery voltage, to alert the user to a low battery.

Other Integrated Hardware

Aside from what might be considered typical peripherals for a microcontroller (UART, SPI, ADC), some AVRs include more specialized peripherals for specific applications.

LCD Driver

In larger models like the ATmega169 (as seen in the AVR Butterfly), an LCD driver is integrated. The LCD driver commandeers several ports of the AVR to drive the column/row connections of a display. One particular trait of liquid crystals that must be taken care of is that no DC bias is put through it. DC bias, or having more electrons passing one way than the other when pumping AC, chemically breaks apart the liquid crystal. The AVR's LCD module uses precise timing to drive pixels forwards and backwards equally.

USB Interface

Main page: Serial Programming/USB

The AT90USB series includes an on-chip USB controller. Some models are "function" only, while others have On-The-Go functionality to act as either a USB host (for interfacing with other slave devices) or as a USB slave (for interfacing with a USB master).

AVRs without built-in USB can use an external chip such as the PDIUSB12, or for a low-speed and minimal functionality device, a firmware-only approach.

Two firmware-only USB drivers are

- obdev, which is available under an Open Source compliant license with some restrictions, and
- USBtiny, which is licensed under the GPL.
- The LUFA library, released under the MIT License, allows the USB-enabled AVR microcontrollers to act as a USB Host, slave or OTG device.[1]

Although these software implementation provide a very cheap way to add USB connectivity, they are limited to low-speed transfers, and tie up quite some AVR resources. Other hardware ICs which translate USB signals to RS-232 (serial) for the AVRs are available, from vendors such as FTDI. These ICs have the advantage of offloading the strenuous task of managing the USB connection with the disadvantage of being limited to the speed of the AVR's serial port. See Embedded Systems/Particular Microprocessors#USB interface for more details on such USB interface chips.

Temperature Sensor

Some newer models have a built in temperature sensor hooked up to the ADC.

AVR Selection

The AVR microcontrollers are divided into three groups:

- tinyAVR
- AVR (Classic AVR)
- megaAVR

The difference between these devices mostly lies in the available features. The tinyAVR microcontrollers are usually devices with lower pin-count or reduced feature set compared to the megaAVRs. All AVR devices have the same basic instruction set and memory organization, so migrating from one device to another AVR is usually trivial.

The classic AVR is mostly EOL'd, and so new designs should use the Mega or Tiny series. Some of the classic AVRs have replacement parts in the mega series, e.g. the AT90S8515 is replaced by the mega8515.

Atmel provides a Parametric Product Table which compares the memory, peripherals, and features available on the entire line of AVRs.

Hardware Design Considerations

Atmel provides the AVR Hardware Design Considerations to assist the hardware designer. This document also shows the standard in-circuit serial programming connector.

AVR development/application boards

Butterfly Demo Board

The AVR Butterfly is a self-contained, battery-powered demonstration board running the ATMEL AVR ATmega169V Microcontroller. The board includes an LCD screen, joystick, speaker, serial port, RTC, flash chip, temperature, light and voltage sensors. The board has a shirt pin on its back and can be worn as a name badge.

The AVR Butterfly comes preloaded with software to demonstrate the capabilities of the microcontroller. Factory firmware can scroll your name, display the sensor readings, and show the time. Also, the AVR Butterfly has a piezo buzzer that can reproduce sound.

The AVR Butterfly demonstrates LCD driving by running a 14-segment, 6 alpha-numeric character display. However, the LCD interface consumes many of the I/O pins.

The Butterfly's ATmega169 CPU is capable of speeds up to 8 MHz, however it is factory set by software to 2 MHz to preserve the button battery life. A pre-installed bootloader program allows the board to be re-programmed with a standard RS-232 serial plug.

Ecros Technology produces a carrier board for the Butterfly which provides a power supply, convenient connections to I/O ports, a DB-9 serial port (with level translator), and a large prototyping area.