Programming ATMEL AVR series microcontrollers

Prof. Prabhat Ranjan
(prabhat_ranjan@daiict.ac.in)
DA-IICT, Gandhinagar

Outline

- AVR advantage
- Features of ATMega32
- Microcontroller Memory
- AVR Registers
- Required software components
- Compiling and linking using avr-gcc
- Converting to Hex format
- Function register access
- Example Programs : (flashing LED, interrupt driven LED, binary counter)

AVR advantages

- Micro-controllers will often allow an optimal solution, combining complex functionality with reduced part count
- ATMEL AVR chips pack lots of power (1 MIPS/MHz, clocks up to 16MHz) and space (up to 128K of flash program memory and 4K of EEPROM and SRAM) at low prices.
- HLL Support, like C, helps increase reuse and reduce turn-around/debug time/headaches.
- In-System Programmable flash--can easily program chips, even while in-circuit.

- Many peripherals: a whole bunch of internal and external interrupt sources and peripherals are available on a wide range of devices (timers, UARTs, ADC, watchdog, etc.).
- 32 registers: The 32 working registers (all directly usable by the ALU) help keep performance snappy, reducing the use of time-consuming RAM access.

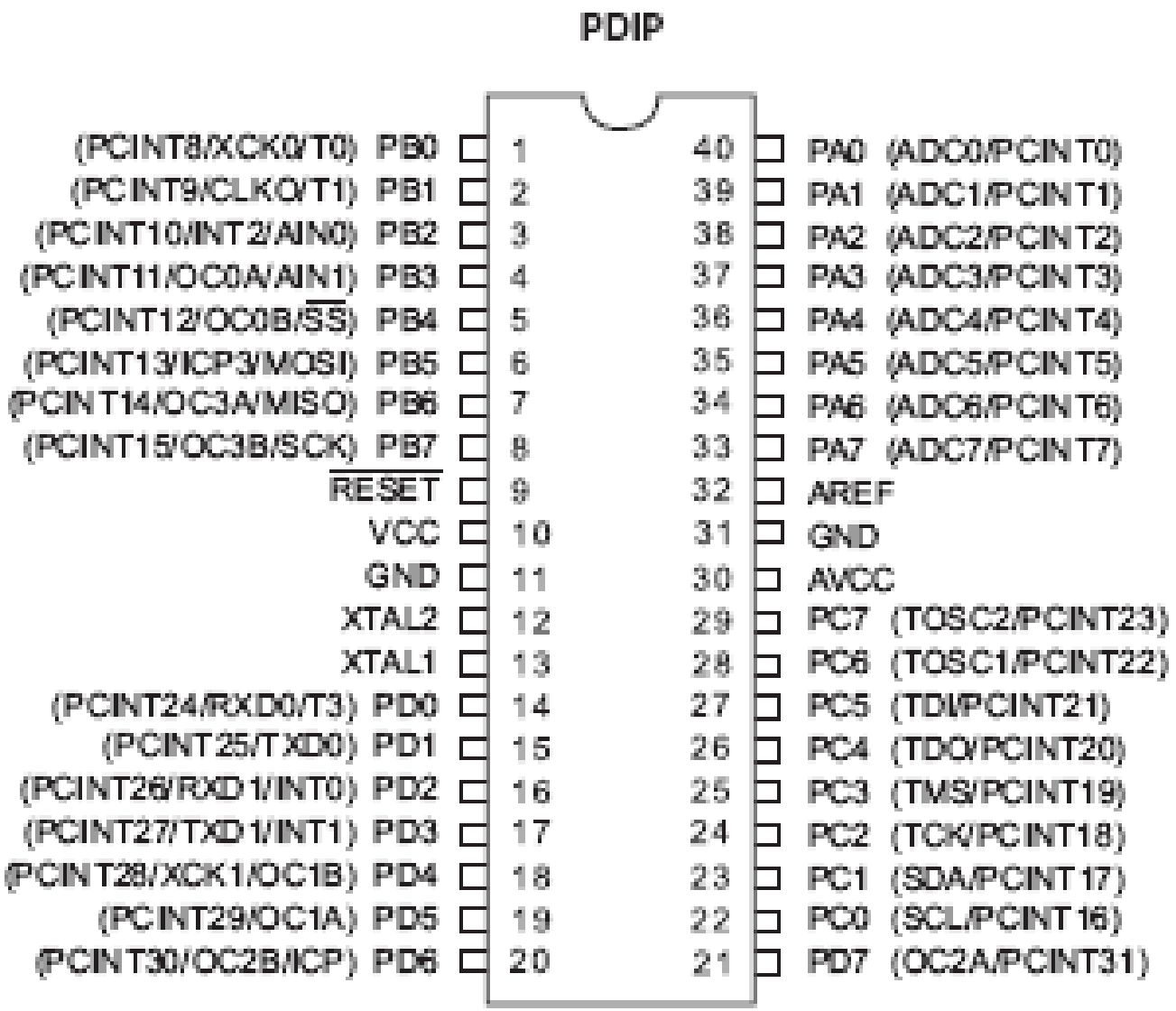
- Internal RC oscillators can be used on many chips to reduce part count further.
- Flexible interrupt module with multiple internal/external interrupt sources.
- Multiple power saving modes.

ATMega32 : Features

- 32Kbytes Flash Program Memory
- 2Kbyte Internal SRAM
- 1024 Bytes EEPROM
- 2 x 8-Bit Timer/Counters and 1 x 16-Bit Timer/Counter
- Four PWM Channels
- 8 Channel 10-Bit ADC
- Programmable Serial USART
- Master / Slave SPI Interface
- Programmable Watchdog Timer
- 32 Programmable I/O Lines
- On-chip Analog Comparator
- Six Sleep Modes for Current Consumption Min-imization
- Programmable Lock for Program Security

Atmega1284

Pinout ATmega1284P

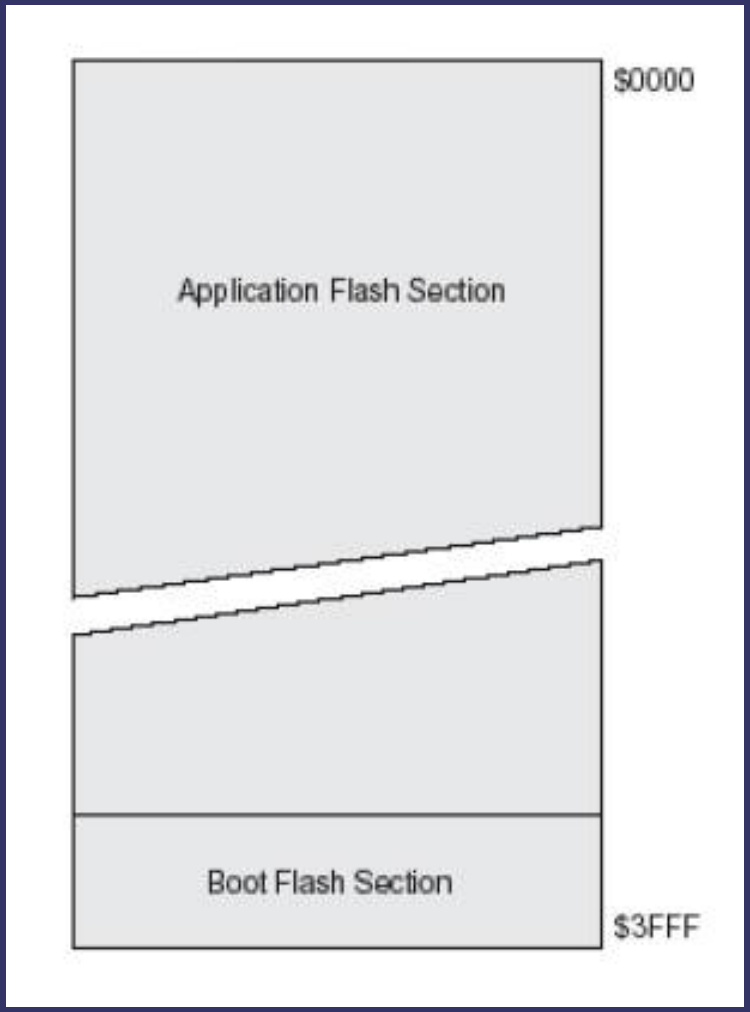


- 128 KB Flash
- 4 KB EEPROM
- 16 KB RAM
- 2 USART
- 1.8 – 5.5 V operation

Microcontroller Memory

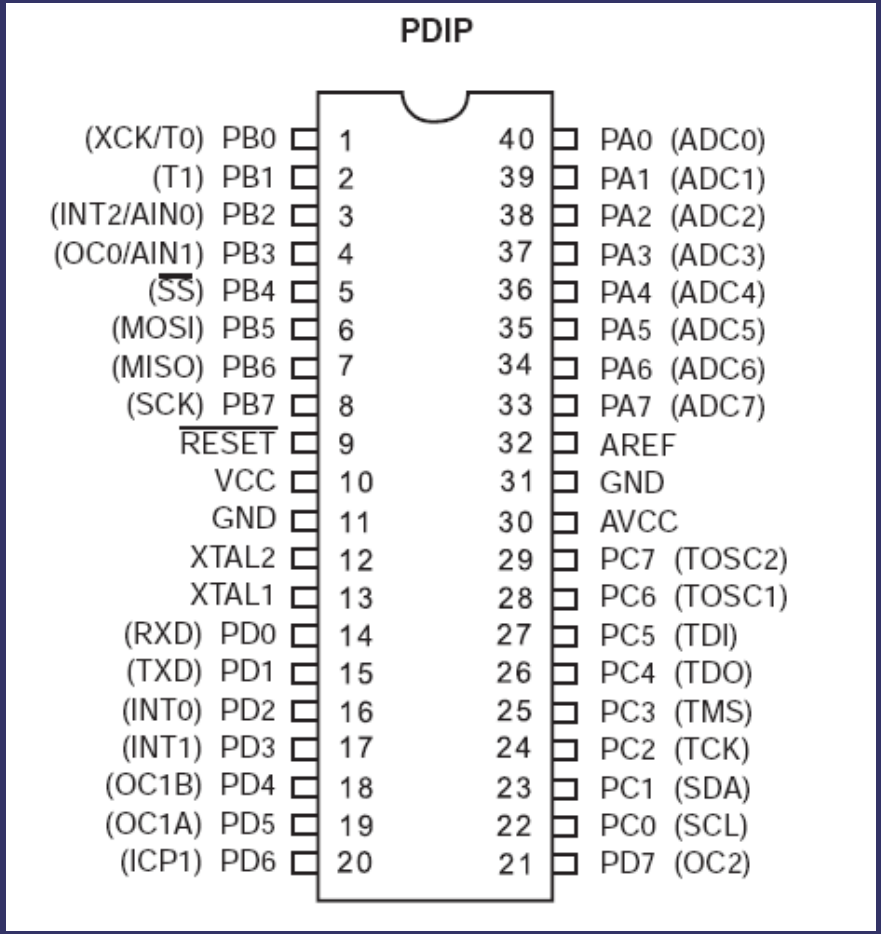
- The ATMega memory consists of three parts:
  - Data memory of SRAM : used for temporary storage of data values
  - Program memory, which is a Flash Memory, that can be rewritten up to 10,000 times
  - Finally the EEPROM memory, which is used for permanent storage of data values or initial parameters for the microcontroller.

Flash Memory



- The boot flash section, is for the boot-loader program, which can be used to program the flash memory by using the on-board UART on the microcontroller

ATMega32 Pinout



- The ATMega32 has 4 ports, these are defined as PORTA, PORTB, PORTC and PORTD.
- Each of the port pins can be used for simple I/O, and in some cases contain a dual function for a peripheral function within the microcontroller.

AVR Registers

- All information in the microcontroller, from the program memory, the timer information, to the state on any of input or output pins, is stored in registers
- The 32 IO pins of the ATMega32 are divided into 4 ports, A, B, C, and D.
- Each port has 3 associated registers.
- For example, for port D, these registers are referred to in C-language by PORTD, PIND, and DDRD

The DDRx Register

- The DDRD register sets the direction of Port D. Each bit of the DDRD register sets the corresponding Port D pin to be either an input or an output.
- A 1 makes the corresponding pin an output, and a 0 makes the corresponding pin an input
  - DDRD=0xF0;
  - Port D : (Pin 0 – 3) - Input
    (Pin 4 -7) - Output

The PORTx Register

- The PORTx register functions differently depending on whether a pin is set to input or output. The simpler case is if a pin is set to output. Then, the PORTC register, for example, controls the value at the physical IO pins on Port C
  - DDRC = 0xFF; //Set all Port C pins to output
  - PORTC = 0xF0; //Set first 4 pins of Port C low and next 4 pins high

The PINx Register

- When a pin is set to input, the PINx register contains the value applied to the pin
  - foo = PIND; //Store value of Port D Pins in foo
- There is also a function available for checking the state of an individual bit, without reading the entire PINx register.
- The function bit_is_set(reg,bit) returns a 1 if the given bit in the given register is set, and a 0 if the bit is cleared
  - bar = bit_is_set(PIND,1);

Required Components

- GCC: The Gnu Compiler Collection, configured and compiled for AVR targets
- BinUtils package, also with AVR specific options enabled
- AVR LibC C library
- A programmer (a hardware programmer, such as our ISP Programmer)
- Help from A programmer or hacker who might enjoy programming micro-controllers and his/her Linux system

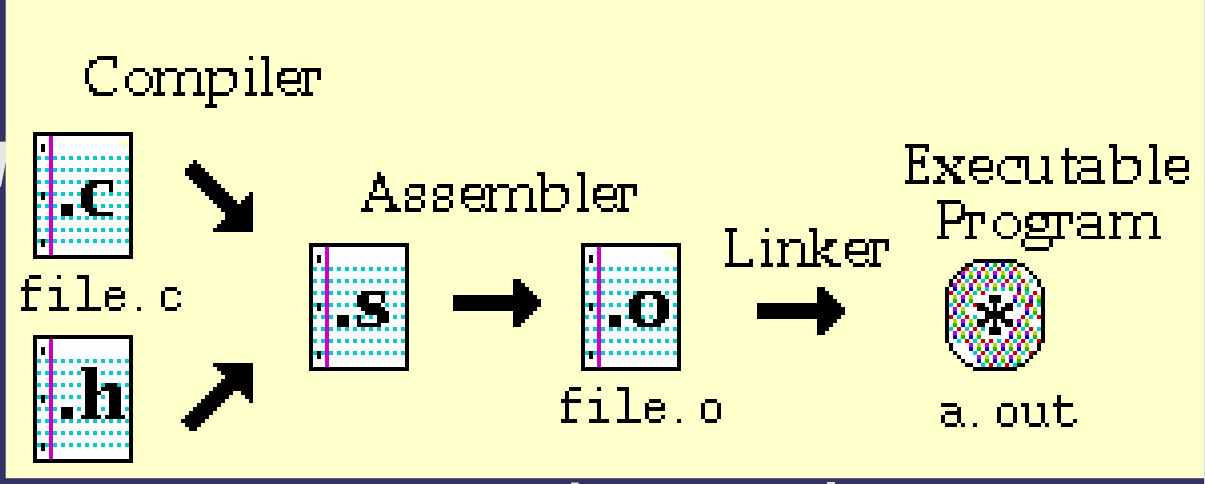
RPM based packages

- avr-binutils-2.14-1
- avr-gcc-3.3.2-1
  - avr-gcc-c++-3.3.2-1(optional)
  - avr-gdb-6.0-1(optional)
- avr-libc-1.0.2-1
- Avr-libc-docs

- http://cdk4avr.sourceforge.net/
- Winavr project

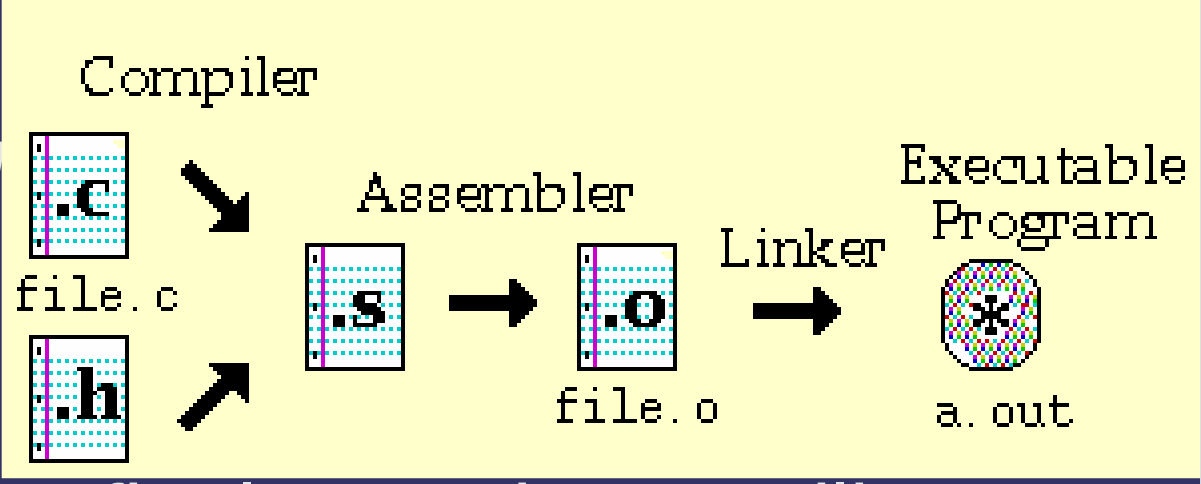
Building Programs

Simple Compilation



- Compiling a small C program requires at least a single .c file, with .h files. There are 3 steps to obtain executable program
- Compiler stage: All C language code in the .c file is converted into a lower-level language called Assembly language; making .s files.
- Assembler stage: The assembly language code is converted into object code which are fragments of code which the computer understands directly. An object code file ends with .o.

Simple Compilation



- Linker stage: The final stage in compiling a program involves linking the object code to code libraries which contain certain "built-in" functions, such as printf. This stage produces an executable program, which is named a.out by default.

Compiling and Linking

- First thing - compile the source
  - avr-gcc -mmcu=atmega32 -c demo.c
- processor type : -mmcu option
- Next link
  - avr-gcc -mmcu=atmega32 -o demo.out demo.o

Generating .hex Files

- Binary of the application: how to get it into the processor
- Extract portions of the binary and save the information into “hex” files. The GNU utility that does this is called “avr-objcopy”
- The ROM contents can be pulled from our project’s binary and put into the file rom.hex using the following command:
  - avr-objcopy -j .text -O ihex demo.out rom.hex

Function Register Access

- Lots of programmer's work involves dealing with I/O and other function registers, often on a bit-by-bit basis
- AVR-LibC provides two methods by which this may be accomplished. One may use specific IO instructions on IO address space, e.g.
  - outb(PORTB, 0xFF);
- Or you can choose to use the symbolic address directly:
  - PORTB = 0xFF;

Manipulating Port Bits/Bytes

- Byte manipulation (equivalent)
  - outb(DDRD,DATA); value = inb(PORTC); (obs)
  - DDRD = DATA; value = PORTC;
- Individual bit manipulation(equivalent)
  - sbi(DDRD, 5); cbi(DDRD,4); (obs)
  - DDRD = _BV(5); DDRD &= ~_BV(4);
  - DDRD = (1 << 5); DDRD &= ~ (1<<4);

A simple program

A program which simply flashes all the LED connected to PORT C!

```
#include <avr/io.h>
void delay_ms(unsigned short ms)
{ put some delay loop}
void main(void)
{
            /*disable Jtag functionality on Port C */
 MCUCSR |= (1<<JTD); MCUCSR |= (1<<JTD);
 DDRC = 0xFF;       /* enable  PORTC as output */
  while (1) {
                /* set output to 0V, LED on */
  PORTC = 0x00;
     delay_ms(500);
                 /* set output to 5V, LED off */
  PORTC = 0xff;
     delay_ms(500);
   }

}
```

MCU Control & Status Reg. - MCUCSR

The MCU Control and Status Register contains control bits for general MCU functions, and provides information on which reset source caused an MCU Reset.

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

- **Bit 7 – JTD: JTAG Interface Disable**

When this bit is zero, the JTAG interface is enabled if the JTAGEN Fuse is programmed. If this bit is one, the JTAG interface is disabled. In order to avoid unintentional disabling or enabling of the JTAG interface, a timed sequence must be followed when changing this bit: The application software must write this bit to the desired value twice within four cycles to change its value.

Interrupt API

- The Avr C-library provides default interrupt routines, which get used unless overridden
- If you wish to provide an interrupt routine for a particular signal you must, in addition to any required AVR setup, create the function using one of the SIGNAL() or INTERRUPT() macros, along with the appropriate signal names

Interrupt API

- There are a number of preset signal names, such as:
  - SIG_ADC (ADC conversion done)
  - SIG_EEPROM_READY
  - SIG_INTERRUPT0..7 (external interrupts 0 to 7)
  - SIG_OVERFLOW0..3 (timer/counter overflow)
  - SIG_UART0_DATA, SIG_UART0_RECV, SIG_UART0_TRANS (UART empty/receive/transmit interrupts)

- To create a interrupt routine, select a macro-signal combination and

```
#include <avr/signal.h>
SIGNAL(SIG_INTERRUPT0)
{
    /* Your interrupt handler routine */
}
```

Example 2 : Interrupt driven

PORT D connected to switches

Pressing one switch puts on all LED
Pressing another one puts off all LED

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

SIGNAL (SIG_INTERRUPT0) /* PD2 */
   {  PORTC=0xFF; /* turn off leds */ }

SIGNAL (SIG_INTERRUPT1) /* PD3 */
   {  PORTC=0x00;  /* turn on leds */ }
```

```
int main( void )
{
   /* PortB - Output (Leds), PortD - Input (Switches) */
   DDRC=0xFF;
   DDRD=0x00;
                /* enable interrupt Int0 and Int1 */
GICR=(1<<INT0)|(1<<INT1);

/* falling edge on Int0/ Int1 generates an interrupt */
MCUCR=(1<<ISC01)|(1<<ISC11);

sei(); // Enable Interrupt
for (;;){}
}
```

General Interrupt Control Reg : GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Interrupt Sense Control : MCUCSR

Table 35. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Example 3 : Binary LED Counter

Use a timer and at every overflow of timer increment a variable

Output this to LED

Clock Scaling

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 42. Clock Select Bit Description (Continued)

CS02	CS01	CS00	Description
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

TIFR Reg : Overflow

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
#include <inttypes.h>
#include <avr/io.h>

uint8_t led;
uint8_t state;

int main( void )
{

DDRC=0xFF;      /* use all pins on PORTC
                   for output */

TCNT0=0;              /* start value of T/C0 */
TCCR0=5;               /* prescale ck/1024 */
```

```
led = 0;

for (;;)   {
do                    /* this while-loop checks the
    overflow bit in the TIFR register */
      state = TIFR & 0x01;
 while (state != 0x01);
       PORTC=~led;
      led++;
      if (led==255)   led=0;

 TIFR=(1<<TOV0);    /* if a 1 is written to the
    TOV0 bit the TOV0 bit will be cleared */
      }
 }
```

Programming Processor

- There are several tools that would burn .hex file in AVR processor
  - uisp – micro-controller in-system programmer
  - avrdude – driver program for simple AVR MCU programmer

UISP

- uisp -v=3 -dprog=stk500 -dserial=/dev/ttyS0 --upload -dpart=auto if=file.hex --erase –verify (STK-500)
- uisp -v=3 -dprog=dapa -dlpt=/dev/parport0 --upload -dpart=auto if=file.hex --erase –verify (kit with parallel port cable : check ?)

avrdude

- avrdude -p atmega32 -P /dev/ttyS0 -c stk500 -U flash:w:file.hex (stk-500)
- avrdude -p atmega32 -P /dev/parport0 -c par -U flash:w:file.hex (Kit)

AVRLib : A great help

- The AVRLib is essentially a collection of functions and macros that make accessing the functionality of the AVR microprocessor easier and more intuitive
- Different from AVR-Libc, which is a subset of C library for AVR

AVRLib : General Use

- Byte Buffering (circular)
- Bit Buffering (linear)
- Printf and other formatted print functions
- VT100 Terminal Output
- Command Line Interface
- FAT16/32 File System (support is read-only for now)
- STX/ETX Packet Protocol
- Fixed-Point Math Library (basic operations only)

AVRLib : Built-In Peripheral Support

- Timer(s)
- Uart(s)
- A/D Converter
- I^2C Master/Slave
- SPI Interface

Device Drivers for External Hard-ware

- Character LCD Modules (HD44780-based)
- I2c EEPROM Memories
- Quadrature Encoders
- RC-Servos (up to 8 channels)
- STA013 MP3 Decoder Chip
- GPS Receivers (via serial port) -- NMEA-0813 Protocol -- Trimble TSIP Pro-tocol
- Graphic LCD Modules -- KS0108/HD61202 Controller -- T6963 Controller -- LCD Fonts and Symbols

Software-Emulated Devices and Interfaces

- I^2c Master (Bit-Bang)
- UART (software-based, timer interrupt driven)
- Pulse Output (arbitrary-frequency continuous/counted square wave)
- Intel-type Memory Bus (Address & Data Buses + nRD,nWR)

AVR Microcontrollers

- AVR 8-bit RISC
- AVR32 – 32 bit CPU
- AT91SAM 32-bit ARM-based Microcontrollers
- Secure Microcontrollers - 8/16 bit and 32 bit
- Military/Aerospace
- Rad Hard ICs

AVR 8-bit RISC

- Automotive AVR
- Lighting AVR
- AVR Z-Link
- megaAVR
- Battery Management AVR
- tinyAVR
- CAN AVR
- USB AVR
- LCD AVR
- XMEGA

PicoPower AVR

- A number of techniques for lower power consumption in sleep and active mode.
- The key elements are:
  - True 1.8V Supply Voltage
  - Minimized Leakage Current
  - Sleeping BOD
  - Ultra Low Power 32 kHz Crystal Oscillator
  - Digital Input Disable Registers
  - Power Reduction Register
  - Clock Gating
  - Flash Sampling
- ATMega164P, ATMega324P, AT-Mega1284P

- “PicoPower enables AVR to achieve the industry's lowest power consumption with 650 nA with a RTC running and 100nA in Power Down sleep”

Thank You