Piconomic Design Atmel AVR Course

Sections:

Tutorial 01 - Port IO

This tutorial demonstrates how to configure and use general purpose I/O pins. The LED lights while the push button is pressed.

Compact C techniques are used to manipulate individual bits of a Peripheral Control Register. It is good programming practice to encapsulate these extensively used (and error prone) pieces of code in macros, as has been demonstrated in the firmware framework:

PORT_LED_O &= ~(1<<BIT_LED_O); // Clear PB6

should rather be replaced with a call to the following macro (defined in "hardware.h"):

#define LED_OFF() BIT_SET_LO(PORT_LED_O,BIT_LED_O)

which uses the following macro (defined in "global.h"):

#define BIT_SET_LO(Port,Bit) {Port &= ~(1<<Bit);}

The register and bit names are defined in "C:\WinAVR\avr\include\avr\iom128.h" (assuming that WinAVR is installed in it's default location).

Tutorial 02 - Timers

This tutorial demonstrates how to configure and use a timer peripheral as a polled timeout/delay function. The buzzer beeps repeatedly for 100 ms with a 1 s delay in between.

One of the goals was to keep the tutorials small, compact and self-contained. Thus a good programming practice to encapsulate a functional block in a separate H and C file was ignored, e.g. all the "TMR_xxx(...)" functions should be kept in a "timer.h" and "timer.c" file.

Tutorial 03 - Interrupts

This tutorial demonstrates how to implement a simple interrupt handler. A timer is configured to generate an interrupt every 250 ms. The LED output pin is toggled every 250 ms in the timer interrupt

A macro **sei()** is called to enable interrupts globally. It sets bit 7 (I) of the SREG register. **cli()** can be called to disable all interrupts.

Refer to "Interrupt Vectors in ATmega128" p.59 of the ATmega128 datasheet to view the list of interrupt vectors.

The vector names are also defined in "C:\WinAVR\avr\include\avr\iom128.h" (assuming that WinAVR is installed in it's default location).

Tutorial 04 - USARTs

This tutorial demonstrates how to configure and use a USART in a polled fashion. USART0 (RS-232 port) is configured to 115200 Baud, 8 Data Bits, No Parity and 1 Stop Bit. The string "AVR\r\n" is sent upon start up. Thereafter all received characters are echoed back to the sender.

The clock rate of the AVR is defined in the Makefile with the macro name "**F_CPU**". It is a good programming practice to calculate values at compile time, in stead of using "magic values", e.g. the value of **UBBR**, which is set with the following macro:

#define UART0_UBBR_VALUE ((F_CPU/(UART0_BAUD<<4))-1)

in stead of

#define UART0_UBBR_VALUE 3 // Magic Value for 115200 Baud at a clock rate of 7.3728 MHz

Tutorial 05 - Printf

This tutorial demonstrates how to direct the AVR-LIBC stdoutstream to send characters using a USART. USART0 (RS-232 port)is configured to 115200B, 8D, NP, 1S. The strings "String in data memory\n" and "String in program memory\n" is sent upon start up.

A custom handler "USART0_iPutChar(...)" is linked to the stdout stream to send a character and inject a "\r" before every "\n".

This example draws attention to the handling of the Harvardarchitecture by the GCC compiler and the AVR-LIBC library. The 8-bit AVRs have a small amount of SRAM (data memory) and therefor it is desirable to store constant strings in FLASH (program memory). The recommended solution is to use the macros in "C:\WinAVR\avr\include\avr\pgmspace.h" and string handling functions with the suffix "_P", e.g. "printf_P(...)" in stead of "printf(...)". The handling has been encapsulated with the following variable argument macro:

#define PRINTF(format, ...) printf_P(PSTR(format), ## __VA_ARGS__)

This functionality is encapsulated in "printf.h" and "printf.c" in the **firmware framework.**

Tutorial 06 - PWM

This tutorial demonstrates how to configure and use a timer peripheral to generate a PWM output. The LED flashes at a rate of 450 Hz with a 1% duty cycle. This will not be observable by the human eye.

An undesirable side-effect effect of the flexibility and power of the timer peripherals is the sheer size of the datasheet documentation. Most times it helps to skip the lengthy description and focus on the actual peripheral register description, e.g. do not start at p.111, but jump directly to "16-bit Timer/Counter Register Description" p.133 of the ATmega128 datasheet.

Tutorial 07 - ADC

This tutorial demonstrates how to configure and use one channel of the ADC converter in a polled fashion. The analogue voltage of AD1 (Pin 60 - PF1) is measured once a second and reported via USART0 @ 115200B, 8D, NP, 1S (RS-232 port). A voltage between +0V to +3.3V may be applied.

The floating point support of the printf library is enabled in the Makefile with:
PRINTF_LIB = $(PRINTF_LIB_FLOAT)

Tutorial 08 - Sleep Mode

This tutorial demonstrates how to use one of the AVR's low power sleep modes. The AVR wakes up every 250 ms, toggles the LED output and returns to sleep mode.

There are various sleep modes, each with increasing levels of power saving. Refer to table 17, "Sleep Mode Select", p.44 of the ATmega128 datasheet to view the list.

The recommended way to implement sleep mode is to use "C:\WinAVR\avr\include\avr\sleep.h" (assuming that WinAVR is installed in it's default location).

Tutorial 09 - EEPROM

This tutorial demonstrates how to write to and read from EEPROM. 0xAB is written to and then read from EEPROM address 0x000C and reported over USART0 @ 115200B, 8D, NP, 1S (RS-232 port). Then 0xCD is written to and then read from EEPROM address 0x000C and reported.

It is recommended to use "C:\WinAVR\avr\include\avr\eeprom.h" (assuming that WinAVR is installed in it's default location).

It is not neccessary to use an explicit EEPROM address. The compiler will allocate a unique address if the EEMEM prefix is used. Here is a usage example:

```
#include <avr/eeprom.h>

EEMEM unsigned short u16Setting_EE;
static unsigned short u16Setting;

int main(void)
{
   // Load setting from EEPROM
   u16Setting = eeprom_read_word(&u16Setting_EE);

   // ...

   // Commit new setting to EEPROM
   eeprom_write_word(&u16Setting_EE, u16Setting);
   eeprom_busy_wait();
}
```

Tutorial 10 - RTC

This tutorial demonstrates how to use a 32.768 kHz crystal and the asynchronous timer 0 as a Real Time Clock (RTC). The LED output is toggled once a second, and time updated (but not used in this tutorial).

There is a common pitfall when flags or variables are updated in an interrupt and the main program flow waits for these flags or variables.

The C keyword "volatile" is used to indicate to the compiler that variables with this prefix are updated asynchronously. The compiler will generate assembler code that fetches the value of the variable each time it is referenced, in stead of optimizing the code by fetching once only. Here is example pseudo code that works correctly:

```
volatile bool bFlag;

ISR(TIMER0_COMP_vect)
{
  // Set flag
  bFlag = TRUE;
}

void Function(void)
{
  // Clear flag
  bFlag = FALSE;

  // Wait for timer interrupt to occur
  while(bFlag == FALSE) {;}

  // Do something after timer interrupt
}
```

The RTC functionality has been encapsulated in "rtc.h" and "rtc.c" of the **firmware framework**.

Tutorial 11 - Assembler

This advanced tutorial demonstrates how to create a complete assembler project in AVR Studio. The LED output pin is toggled every 250 ms in the timer interrupt. It is a recreation of "Tutorial 03 - Interrupts" in assembler.

The aim of the project is to simulate the code in AVR Studio. The project creates a "Assembler.hex" file and this needs to be converted to "Assembler.bin", before it can be uploaded via the XMODEM-CRC bootloader.

Here are the steps to build and simulate the project:

1. Open the project in AVR Studio: "Project > Open Project" and select the AVR Studio project, "..\Tutorials\12 Assembler\Assembler.aps"

2. Build and run the project: "Build > Build and Run"

3. Use the "Debug" menu to start and stop debugging, single-stepping, etc.

Tutorial 12 - Power Management

This advanced tutorial demonstrates how to use the asynchronous timer 0 and the 32.768 kHz crystal to achieve a deeper sleep mode called "POWER-SAVE". The AVR wakes up every 50 ms and "mirrors" the state of the push button to the LED. The AVR will also wake up asynchronously on a rising or falling edge interrupt on PD0 (INT0/SCL) and output the state of PD0 to PD1 (INT1/SDA). This feature can be used to measure the responsiveness of the AVR using a signal generator and an oscilloscope.

The main 7.3728 MHz crystal is disabled when the AVR enters "POWER-SAVE" sleep mode. When the asynchronous timer 0 interrupt or external interrupt 0 wakes up the AVR, the crystal takes some time to oscillate at the proper frequency.

A caveat of the AVR is that the main crystal will not be disabled while the JTAG interface is enabled. It can be disabled by programming the AVR fuse bits, or using a specific sequence of code, which is included in the tutorial. This will also be necessary when ADC4-7 of the ADC converter is needed (which is shared with the JTAG interface).

Tutorial 13 - External Memory

NOTE: The Piconomic Design AVR Development Board does not have external SRAM and this tutorial will not execute correctly without it. It is included here to serve as a starting point for designs with external SRAM.

Here is a schematic example of how to interface to 32K external SRAM.

This tutorial demonstrates how to enable the external memory interface and redirect the heap to external memory. A 64-byte block of external SRAM is filled with a test pattern (first zero's, then incrementing values) and reported via USART0 @ 115200B, 8D, NP, 1S (RS-232 port) .

The linker must be informed that external memory must be used for the heap (the default location is internal SRAM, between the uninitialized variables section and the downwards growing stack). The following line in the Makefile accomplishes this:

EXTMEMOPTS = -Wl,--defsym=__heap_start=0x801100,--defsym=__heap_end=0x8090ff

Refer to section 8.5 "Using Malloc" of the AVR-LIBC Reference Manual for a detailed explanation.

It is recommended to place the function "XMEM_vEnable(void)" in the ".init1" section to enable external memory early during the start up of the application.

History Log

2007-02-20

Upgraded all Makefiles to WinAVR 2007-01-22 template.
Tested all source code with WinAVR 2007-01-22 and AVR Studio 4.13 build 524

Changes to Firmware Framework:

- Improved timer module. Note that legacy code will be broken.
- Renamed "debug.h" to "dbg.h".
- Improved RTC module.

Updated AVR Studio course pictures to version 4.13 build 524

2007-02-15

Fixed link in Tutorial 13 - External Memory

2006-11-20

Added errata for AB111-4 PCB.

2006-11-15

Added Google ads to subsidize hosting cost of AVR course.

Updated Firmware Framework. Changes:

- Added debug trace module (debug.h).
- Fixed bugs in TWI (I2C compatible) module.
- Added support for REPEATED START condition in TWI module.

Updated Tutorial 09 - EEPROM with EEMEM example.

2006-11-03

Added reviews page.

2006-10-14
Moved ZIP files to "/download" directory to conserve crawler bandwidth with "/robots.txt" file

2006-10-04

Assigned explicit names to HTML named anchors.
Added page counter.

2006-10-03

Schematic, PCB and BOM updated to AB111-4. Improvements:

- Changed all 0603 discretes to 0805 SMD footprint.
- Moved electrolytic capacitors from edge of board to accommodate v-scored PCB separator.
- Increased hole size of RS-485 terminal block for easier insertion during assembly.
- Added sponsorship logos of Atmel, ELPROM and W.H. Circuits to bottom silkscreen.

Switched to AVR Studio and the AVR GCC plugin as the preferred development environment.
Generated AVR Studio project files for bootloader, tutorials and firmware framework.
Added "template.h" and "template.c" to firmware framework.
Added revision suffix to all downloadable files.

2006-09-18

Added link to order page.
Schematic, PCB and BOM updated to AB111-3. Improvements:

- Added PNP transistor to drive buzzer.

2006-09-08

Listed source code modules in firmware framework.

2006-09-07

Official release of version 2 of development board.
Updated web page.
Updated Bootloader, Firmware Framework and Tutorials source code.

2006-08-21

Initial draft release.