Detailed Explanation of `BlinkingLEDUsingTimer0.asm` Code

In this note, I will try to give an explanation of the code `BlinkingLEDUsingTimer0.asm` assembly code with references to sections in the huge ATmega328P datasheet

`Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf`

that should be in your Pi in the directory `/home/pi/Programming/Assembly/ATmega328P_Documentation/` .

Walking through the assembly source code

The `BlinkingLEDUsingTimer0.asm` code is in the directory `/home/pi/Programming/Assembly/BlinkingLED/ and contains the following:`

```
.NOLIST
.INCLUDE "../m328Pdef.inc"
.LIST
.org    0x0000
; Have timer/counter-0 cause a toggle of the state of OC0A (PIND6) each time the count
;    becomes 0.  Connect PD6 to B1 and jumper B1 output to see blinking.
ldi     r16,(0<<COM0A1)|(1<<COM0A0)|(0<<COM0B1)|(0<<COM0B0)|(0<<WGM01)|(0<<WGM00)
out     TCCR0A,r16

sbi     DDRD,DDD6               ; Prepare PIND6 to be an output to drive LED.

; System clock, by default, is its calibrated RC oscillator operating at 8 MHz.
; The pre-scaler, by default, is set to divide by 8 so ClkIO is 1 uS.
; The following sets the timer/counter-0 divider to divide by 1024 so it ticks every 1024 uS
; Since this is an 8-bit counter it will rollover every 256*1024 uS and toggle the LED.
; The LED will then have a cycle time of 512*1024 us = 524288 uS
; Note: If CS00 is set to 1 instead, the blinking cycle time will be 262144 uS.
ldi     r16,(0<<FOC0A)|(0<<FOC0B)|(0<<WGM02)|(1<<CS02)|(0<<CS01)|(1<<CS00)
out     TCCR0B,r16

; Interrupts will be disabled by default

; To minimize power consumption, disable power to all subsystems except the timer/counter-0.
; Note: Default is 0x00, all subsystems powered.
ldi     r16,(1<<PRTWI)|(1<<PRTIM2)|(0<<PRTIM0)|(1<<PRTIM1)|(1<<PRSPI)|(1<<PRUSART0)|(1<<PRADC)
sts     PRR,r16

ldi     r16,(0<<SM2)|(0<<SM1)|(0<<SM0)|(1<<SE)    ; Go to ADC low-noise sleep
out     SMCR,r16
SLEEP                   ; Since interrupts are disabled, it will never wake up
```

The first 4 lines start with a period and are directives to the avra assembler as explained in section 4.5 of the `AVRAUsersGuide-doc1022.pdf` found in the `/home/pi/Programming/Assembly/Avra_Compiler/` directory. This file only has Section 4 of a larger document. The complete manual is at

*www.**atmel**.com/images/**Atmel**-0856-AVR-Instruction-Set-Manual.pdf*

which should be downloaded and studied.

Normally, the compiler is run by simply doing `avra BlinkingLEDUsingTimer0.asm` which produces the following files:

```
BlinkingLEDUsingTimer0.cof
BlinkingLEDUsingTimer0.eep.hex
BlinkingLEDUsingTimer0.hex
BlinkingLEDUsingTimer0.obj
```

The .LIST directive on the 3rd line of the assembly code will produce another file, but only if the compiler is invoked with the -l <filename> option. For example,

`avra -l BlinkingLEDUsingTimer0.lst BlinkingLEDUsingTimer0.asm`

which produces the file `BlinkingLEDUsingTimer0.lst` containing:

```
AVRA   Ver. 1.3.0 BlinkingLEDUsingTimer0.asm Sun Jun 21 13:32:25 2015


          .LIST
          .org    0x0000
         ; Have timer/counter-0 cause a toggle of the state of OC0A (PIND6) each time the count
         ;    becomes 0.  Connect PD6 to B1 and jumper B1 output to see blinking.
C:000000 e400      ldi     r16,(0<<COM0A1)|(1<<COM0A0)|(0<<COM0B1)|(0<<COM0B0)|(0<<WGM01)|(0<<WGM00)
C:000001 bd04      out     TCCR0A,r16
```

```
C:000002 9a56      sbi    DDRD,DDD6               ; Prepare PIND6 to be an output to drive LED.

         ; System clock, by default, is its calibrated RC oscillator operating at 8 MHz.
         ; The pre-scaler, by default, is set to divide by 8 so ClkIO is 1 uS.
         ; The following sets the timer/counter-0 divider to divide by 1024 so it ticks every 1024 uS
         ; Since this is an 8-bit counter it will rollover every 256*1024 uS and toggle the LED.
         ; The LED will then have a cycle time of 512*1024 us = 524288 uS
         ; Note: If CS00 is set to 1 instead, the blinking cycle time will be 262144 uS.
C:000003 e005      ldi    r16,(0<<FOC0A)|(0<<FOC0B)|(0<<WGM02)|(1<<CS02)|(0<<CS01)|(1<<CS00)
C:000004 bd05      out    TCCR0B,r16

         ; Interrupts will be disabled by default

         ; To minimize power consumption, disable power to all subsystems except the timer/counter-0.
         ; Note: Default is 0x00, all subsystems powered.
C:000005 ec0f      ldi     r16,(1<<PRTWI)|(1<<PRTIM2)|(0<<PRTIM0)|(1<<PRTIM1)|(1<<PRSPI)|(1<<PRUSART0)|(1<<PRADC)
C:000006 9300 0064 sts     PRR,r16

C:000008 e001      ldi    r16,(0<<SM2)|(0<<SM1)|(0<<SM0)|(1<<SE)    ; Go to ADC low-noise sleep
C:000009 bf03      out    SMCR,r16
C:00000a 9588      SLEEP                   ; Since interrupts are disabled, it will never wake up


Segment usage:
   Code      :        11 words (22 bytes)
   Data      :         0 bytes
   EEPROM    :         0 bytes

Assembly completed with no errors.
```

The `.NOLIST` directive turns off listing even when the compiler is invoked with the `-l` <filename> option. This is used in our code so that our listing is not bloated with all the code brought in by the directive that follows it, `.INCLUDE "../m328Pdef.inc"` . The `.INCLUDE` "<filename>" directive brings in all the code in the specified file. In this case the file `m328Pdef.inc` located in the directory above our current directory. Use your `vim` editor to see that code without the possibility of accidentally changing it by doing `view ../m328Pdef.inc` . It has 977 lines of instructions and comments that tell the compiler human-readable definitions of registers and bits available to the programmer. We do not want the listing to include all this so it is preceded by a `.NOLIST` directive. (In that file are seven `#pragma` directives that I have commented out by placing a semicolon at the start of those lines. They cause annoying warnings during compliation with the avra complier we are using.)

The next line in our code `.org 0000` tells the avra compiler at what memory address in the microprocessor the following code should be loaded.

Finally, after two lines of comments, we get to some real code, but to understand how to use the code, we must understand that not all memory locations are created equal. This is explained in Section 8 of the ATmega328P datasheet. In short, there are general-purpose registers (32 B), flash memory (32 kB), EEPROM memory (1 kB), and RAM (2 kB). The general-purpose registers are very fast, but can only be used by certain instructions and with certain limitations, flash memory is retained when the processor is powered down. It is where programs are program constants are stored. EEPROM memory is also retained, but is more difficult to change and is where certain configuration data (called “fuses”) are held. An `ERASE` command erases flash memory, but not EEPROM memory. RAM is temporary memory where changing data might be held. It is lost when the processor is powered down.

Section 37 (pages 626-628) of the ATmega328P datasheet lists the assembly program mnemonic codes. The abbreviations used in that table are fully explained in the long Atmel Instruction Set Manual referred to above. Here, I will just give a short summary.

In that table, the second column and the fourth column use the following abbreviations:

- R_d Registers `R0` through `R31`
- R_r Registers `R16` through `R31` only
- R_{d1} Double Registers (`R24`, `R25`), (`R26`, `R27`), (`R28`, `R29`), and (`R30`, `R31`)
- `k` A 12-bit constant (value 0-4095)
- `K` A 6-bit constant (value 0-63)
- `b` A 3-bit constant (value 0-7)
- `q` A 6-bit signed displacement (value 0-63)
- `(I/O)P` Input/Output ports 0-31
- `X` Value in double register (`R24`,`R25`)

(X)	Value at address in double register (R24,R25) for indirect addressing
Y	Value in double register (R26,R27)
(Y)	Value at address in double register (R26,R27) for indirect addressing
Z	Value in double register (R28,R29)
(Z)	Value at address in double register (R28,R29) for indirect addressing or accessing program memory
s	Some particular flag bit (value 0-7, see below)
SREG	Register holding the following flag bits:
C	Carry flag
Z	Zero flag
N	Negative flag
V	Overflow flag
S	Sign bit
H	Half-carry flag
T	“T” bit
I	Interrupts enabled flag
PC	Program counter (by the end of an instruction, PC points to the next instruction)
STACK	The addressed pointed to by the Z register pair when used as a stack pointer.
SP	Stack pointer

Our first two lines of real code are

```
ldi     r16,(0<<COM0A1)|(1<<COM0A0)|(0<<COM0B1)|(0<<COM0B0)|(0<<WGM01)|(0<<WGM00)
out     TCCR0A,r16
```

ldi stands for the load immediate instruction found on page 627 of the ATmega328P datasheet. r16 represents general-purpose register number 16. COM0A1, COM0A0, COM0B1, COM0B0, WGM01, and WGM00 represent bit values explained in Section 15 of the ATmega328P datasheet, specifically Section 15.9.1 (pages 105-107). The bit-or'ed combination shown ends up producing a value of 64 that is placed in r16. The purpose for this is explained later.

The out instruction then puts that value into the TCCR0A register which we can see from page 624 of the ATmega328P datasheet is at I/O port 0x24 accessed by memory address 0x44.

The next instruction configures PIND6 for output by doing:

```
sbi     DDRD,DDD6               ; Prepare PIND6 to be an output to drive LED.
```

On page 625 of the ATmega328P datasheet at I/O port 0x0A (memory address 0x2A), you will find register DDRD (data direction register D) and bits DDD0 through DDD7. These set the data directions for PIND0 through PIND7. Here PIND6 is set for output by the set bit immediate (sbi) instruction.

After a couple more comments, we find two more instructions

```
ldi     r16,(1<<PRTWI)|(1<<PRTIM2)|(0<<PRTIM0)|(1<<PRTIM1)|(1<<PRSPI)|(1<<PRUSART0)|(1<<PRADC)
sts     PRR,r16
```

These turn power on to the timer0 circuits. A “0” means apply power, and a “1” means turn off. See page 624 of the ATmega328P datasheet where at address 0x64 is the PRR register and the PRTWI, PRTIM2, PRTIM0 PRTIM1, PRSPI, PRUSART0, PRADC bits that might be set in it. At the far right of that line is the page number, 41, in the manual where there is more explanation. These instructions put the number 15+128+64=207 into the memory location 0x64 which turns on only timer 0.

Finally, the program ends with the lines:

```
ldi     r16,(0<<SM2)|(0<<SM1)|(0<<SM0)|(1<<SE)    ; Go to ADC low-noise sleep
out     SMCR,r16
SLEEP                   ; Since interrupts are disabled, it will never wake up
```

You can find the sleep control register SMCR at location I/O port 0x33 (memory location 0x53) and at the end of that line a reference to page 39 of the ATmega328P datasheet. Going to that page and reading section 10.4, we find that the bit set in these instructions set the sleep mode to “ADC Noise Reduction Mode.” The following SLEEP instruction puts

it to sleep.

So why does the LED connected to pin `OC0A` (`PIND6`) blink? To understand the ports, we need to study Section 14 of the ATmega328P datasheet, in particular section 14.3.3, Table 14-9 (page 89) and page 90 near the top which explains that "`AIN0/OC0A/PCINT22` – Port D, Bit 6" when configured as output is an external output for the Timer/Counter0 Compare Match A. Indeed, we have already set `PIND6` for output, but what value is being compared with the value of timer0?

Section 15.5 starting on page 96 explains the compare function. We find that it is comparing the count to the value in `OCR0A` and `OCR0B` listed on page 624 at I/O ports 0x27 and 0x28, and detailed on page 109 of the ATmega328P datasheet. There, we see that these have default values of 0. So each time the timer count `TCNT0` becomes 0, the output of `PIND6` is toggled from "0" to "1" or from "1" to "0" as is explained in Table 15.2 (Section 15.9.1, page 105) when `COM0A1` is set to "0" and `COM0A0` is set to "1" as was done earlier. All this assumes that the "Waveform Generator Mode Bit" for timer0 is set for "normal" mode. For that, `WGM00`, `WGM01` and `WGM02` bits must be all set to "0". The earlier instruction that set `TCCR0A`, set `WGM01` and `WGM00` to "0", but we did not explicitly set `WGM02` to zero in `TCCR0B`. It defaulted to "0" as can be seen in Section 15.9.2 (page 108, top). Depending on default values can lead to future subtle bugs when the code is modified.

Moving the assembled code to the ATmega328P processor

Once the assembly program has been converted to object code, `BlinkingLEDUsingTimer0.obj`, that object code data must be passed on to the ATmega328P processor. This is done using the python program `uploadingViaSPI-Python3.py` which parses the `.obj` file and uses instructions provided in Section 28.8 (pages 299-302) of the ATmega328P datasheet. The RESET line is pulled-low by the Pi, loading instructions are sent to the ATmega328P via the SPI protocol, and when the RESET line is released, the processor starts executing the program. Since the program is in Flash memory, it will remember the program even when the power has been cycled.

Setting the "fuses" in EPROM

Programming the "fuses" in EEPROM is described in Section 28.2 (pages 286-289). The default for the fuses as supplied by the manufacturer is adequate for our purposes, but if not we would need to write a variation of the uploading program to query or alter the EEPROM. For example, we might want to change the clock settings.

This exercise should drive home the need to fully understand the first 319 pages of the ATmega328P data sheet and to be familiar with the instruction and register tables on its pages 622-628. There are lots of gotchas and subtle interactions between the various processor function settings.