FPSLIC Training Series (AVR Assembly based designs)

Software Requirements

- System Designer
  - Atmel AVR Macro Assembler
  - Atmel AVR Studio
- CPS

Hardware Requirements

- ATSTK94 Starter Kit

LAB 1: AVR Assembler and Programming Utility

The goal of LAB1 is to familiarize yourself with the *AVR* Macro Assembler (included with System Designer), the *FPSLIC* Programming Utility (***CPS***), and the ATSTK94 FPSLIC Starter Kit. This LAB will teach you how to assemble and download a program to the Starter Kit using the Atmel *FPSLIC* Tools.

Design Overview-LEDTEST.ASM

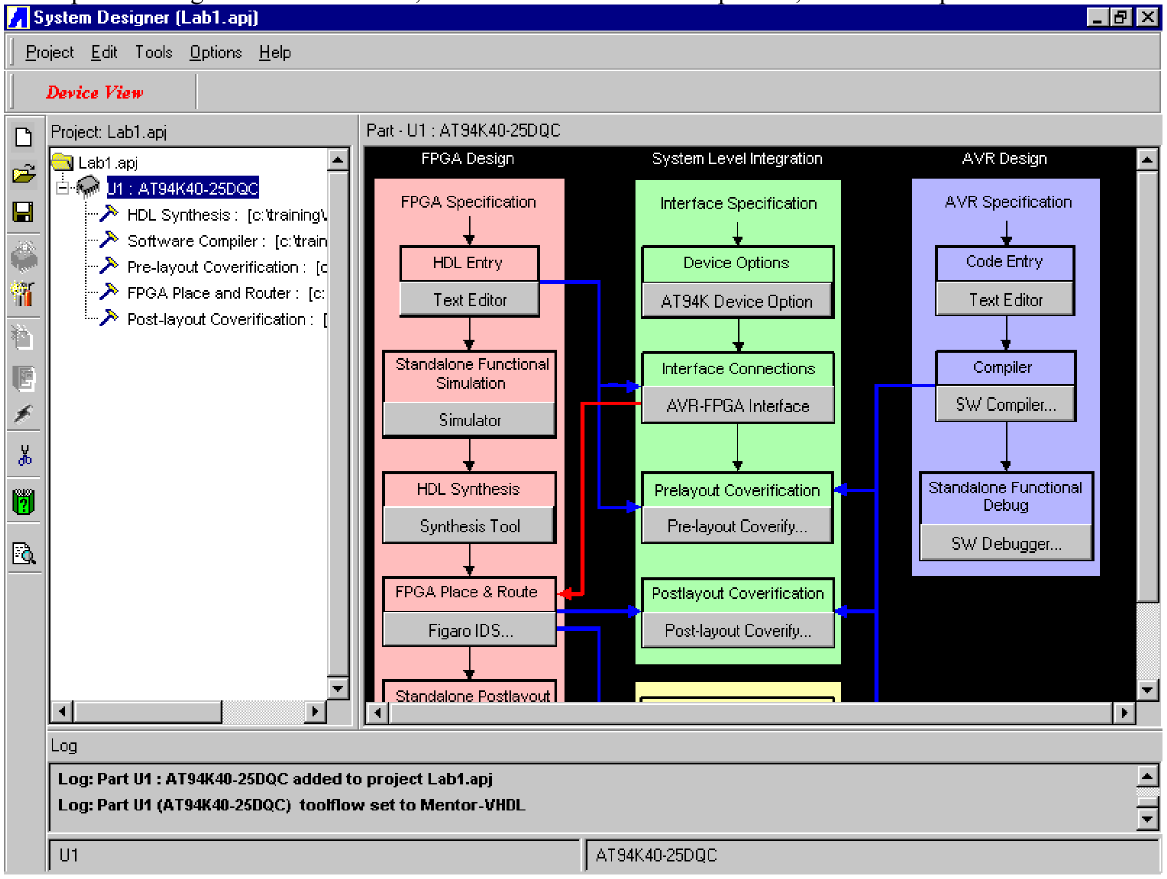
The `LEDTEST.ASM` program **(Figure 1**) is an assembly language program, which uses the LEDs (LED1-LED8) on the Starter Kit board to display a binary counter pattern. This program configures `PORTD` (AT94K) as an output and uses a variable, "`Temp`", to store the current count.

```
.include "at94kdef.inc"         ; LED Test

.def    Temp    =r16            ;temporary register
.def    Delay   =r17
.def    Delay2  =r18

;***** Initialization

RESET:
        ldi     temp,low(0x0FFF)
        out     SPL,temp
        ldi     temp,high(0x0FFF)
        out     SPH,temp        ;init Stack Pointer
        ser     Temp
        out     DDRD,Temp       ;PORTD = all outputs
        clr     Temp            ;Clear Temp = $00

;**** Test input/output
loop:
        out     PORTD,Temp      ;output data to PORTD
        inc     Temp

;**** Now wait a while to make LED changes visible.
DLY:
        dec     Delay
        brne    DLY
        dec     Delay2
        brne    DLY
        rjmp    loop            ;repeat loop endlessly
```

Figure 1: `LEDTEST.ASM`

Creating a New Project in System Designer

1. Go to Start > Programs > Atmel > System Designer3.0 to open System Designer .
2. From the "Project" menu select "New…"
3. Select "New Project Wizard". This will open the New Project Wizard, which allows you to choose your project directory, select which part you want to design with and configure the design tool flow.
4. Press "Next >"
5. Set-up the Design Directory – `C:\TRAINING\AVR\LAB1`
6. Setup the Design File Name – `LAB1`
7. Press "Next >"
8. In the "New Project Wizard Parts Dialog" select the AT94K40-25DQC, this is the 208-pin TQFP device on the ATSTK94.
9. Press "Next >"
10. In the "New Project Wizard Tool Flow Dialog select "Mentor – VHDL"
11. Press "Next >"
12. It is possible to add multiple parts to work on using the New Project Wizard Add More Parts Dialog, but for this lab session we will focus on a single AT94K device.
13. Press "Next >"
14. Press "Finish" on the New Project Wizard Congratulations Dialog.
15. Upon clicking the "Finish" button, the "Device View" will be present, click on the part to enter the "Design Flow View".



```
Log: Part U1 : AT94K40-25DQC added to project Lab1.apj
Log: Part U1 (AT94K40-25DQC) toolflow set to Mentor-VHDL
```

Figure 1: System Designer Design Flow

The Design Flow view shows the steps required in doing a design for FPSLIC, using System Designer and Co-verification. The arrows on the diagram show dependencies between the steps. So for example if you want to run Pre-layout Co-verification, you have to do HDL Entry, Code Entry and compilation, and then define the interface between the AVR and FPGA first.

In these labs we only concentrate on AVR part of FPSLIC (AVR Standalone) not on FPGA/Co-verification part of it. To do an AVR design entry we can either follow the purple path or use the included editor in ***WAVRASM*** for source code entry; we will choose the second option.

Writing Assembly in WAVRASM

The included Assembler in System Designer translates assembly source code into object code. The generated object code can then be used by *AVR* Studio to simulate or emulate the behavior of the AVR Microcontroller. The Assembler generates fixed code allocations; consequently, no linking is necessary.

AVR Assembler Exercises

1. Open AVR Macro Assembler program by using the "Software Compiler" button in System Designer.
2. Open the Assembly file for editing by choosing File -> Open and navigating to the C:\TRAINING\AVR\LAB1 directory and select `LEDTEST.ASM` (See **Figure 2**).
3. Assemble the file by pressing "Assemble".
4. Answer the questions below, and close the assembler.

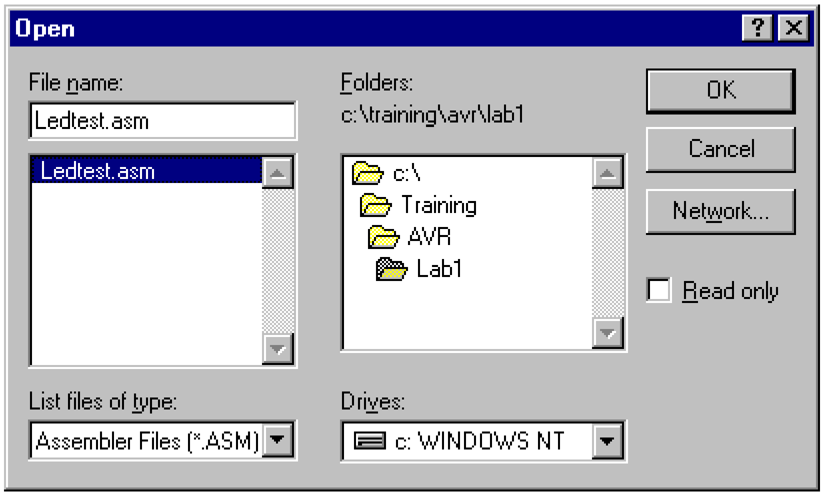


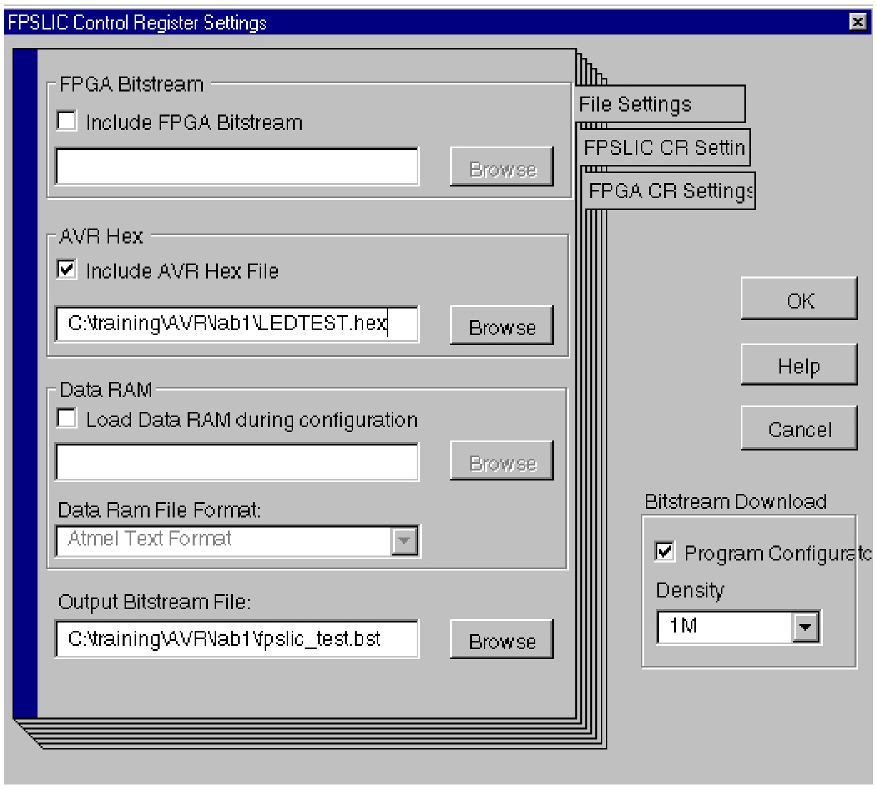
Figure 2: *WAVRASM* Open File Dialog

AVR Assembler Questions

1. How many bytes of Program Memory does this program require? Answer: ___________

Programming the Device

1. Press the “Device Programming” Button in System Designer
2. A window similar to **Figure 3** will open


FPSLIC Control Register Settings

FPGA Bitstream
☐ Include FPGA Bitstream

AVR Hex
☑ Include AVR Hex File
C:\training\AVR\lab1\LEDTEST.hex

Data RAM
☐ Load Data RAM during configuration

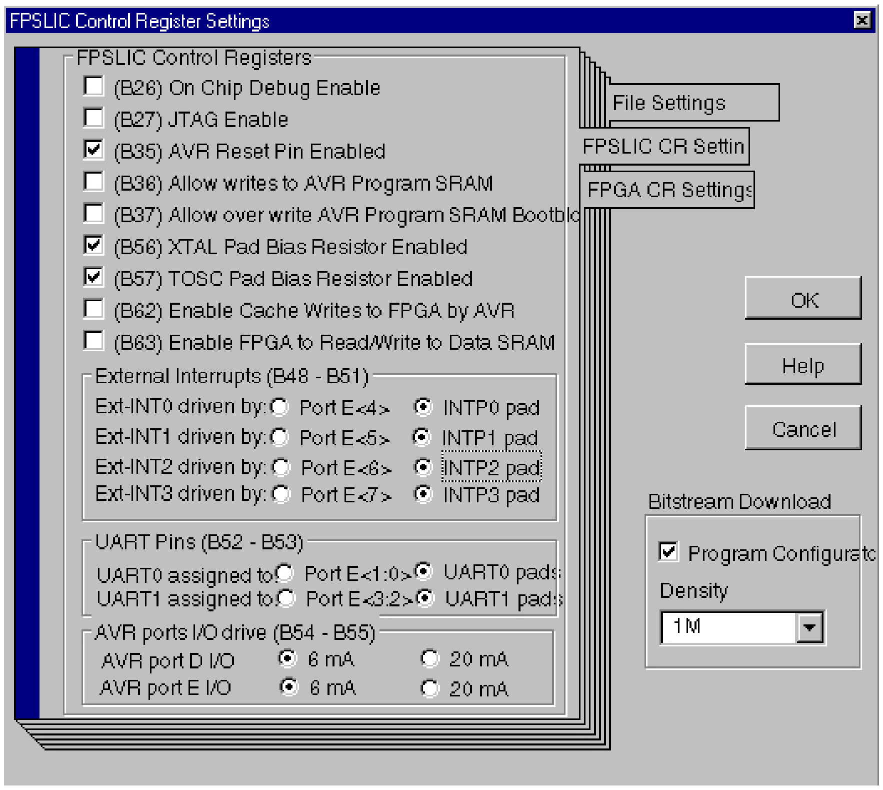
Data Ram File Format: Atmel Text Format

Output Bitstream File: C:\training\AVR\lab1\fpslic_test.bst

Bitstream Download
☑ Program Configuratio
Density: 1M

Figure 3: Bitstream File Settings Window

3. Uncheck the “Include FPGA Bitstream” Option
4. Under “AVR Hex File” press the “Browse” Button, navigate to and select the C:\TRAINING\AVR\LAB1\LEDTEST.hex File
5. Press the “FPSLIC Control Register Settings” Tab and confirm that the options are set according to **Figure 4**.


FPSLIC Control Register Settings

FPSLIC Control Registers

- ☐ (B26) On Chip Debug Enable
- ☐ (B27) JTAG Enable
- ☑ (B35) AVR Reset Pin Enabled
- ☐ (B36) Allow writes to AVR Program SRAM
- ☐ (B37) Allow over write AVR Program SRAM Bootblo
- ☑ (B56) XTAL Pad Bias Resistor Enabled
- ☑ (B57) TOSC Pad Bias Resistor Enabled
- ☐ (B62) Enable Cache Writes to FPGA by AVR
- ☐ (B63) Enable FPGA to Read/Write to Data SRAM

External Interrupts (B48 - B51)

- Ext-INT0 driven by: Port E<4> [✓ INTP0 pad]
- Ext-INT1 driven by: Port E<5> [✓ INTP1 pad]
- Ext-INT2 driven by: Port E<6> [✓ INTP2 pad]
- Ext-INT3 driven by: Port E<7> [✓ INTP3 pad]

UART Pins (B52 - B53)

- UART0 assigned to: Port E<1:0> [✓ UART0 pad]
- UART1 assigned to: Port E<3:2> [✓ UART1 pad]

AVR ports I/O drive (B54 - B55)

- AVR port D I/O [✓ 6 mA] 20 mA
- AVR port E I/O [✓ 6 mA] 20 mA

Bitstream Download

- ☑ Program Configurato
- Density: 1M

Figure 4: Bitstream Control Register Settings

6. Before pressing "OK", some hardware connections need to be performed:
   - Connect the 25-pin parallel cable to the 25-pin Male connector of the ATDH2225 download cable, the 10-pin female header plugs into the 10-pin male header (**J1**) on the ATSTK94 Board.
   - Connect the power supply from an AC outlet to the 9V DC connector (**P3**) on the ATSTK94 Board
   - Make sure to set the **Jumpers** that are located in between the **LEDs** and **Switches** to the AVR (Switch side)
   - Adjust **SW10** to **PROG**
   - Adjust **SW14** to **ON** position
7. Press "OK" on the FPSLIC Control Settings dialog. CPS will automatically open and download the design to the device.
   - Check if AT94K Configuration has been successful:
   - Adjust SW10 to RUN Position
   - Press Reset switch (SW12) on the right hand board edge
   - You should see the LEDs count-up in a binary pattern

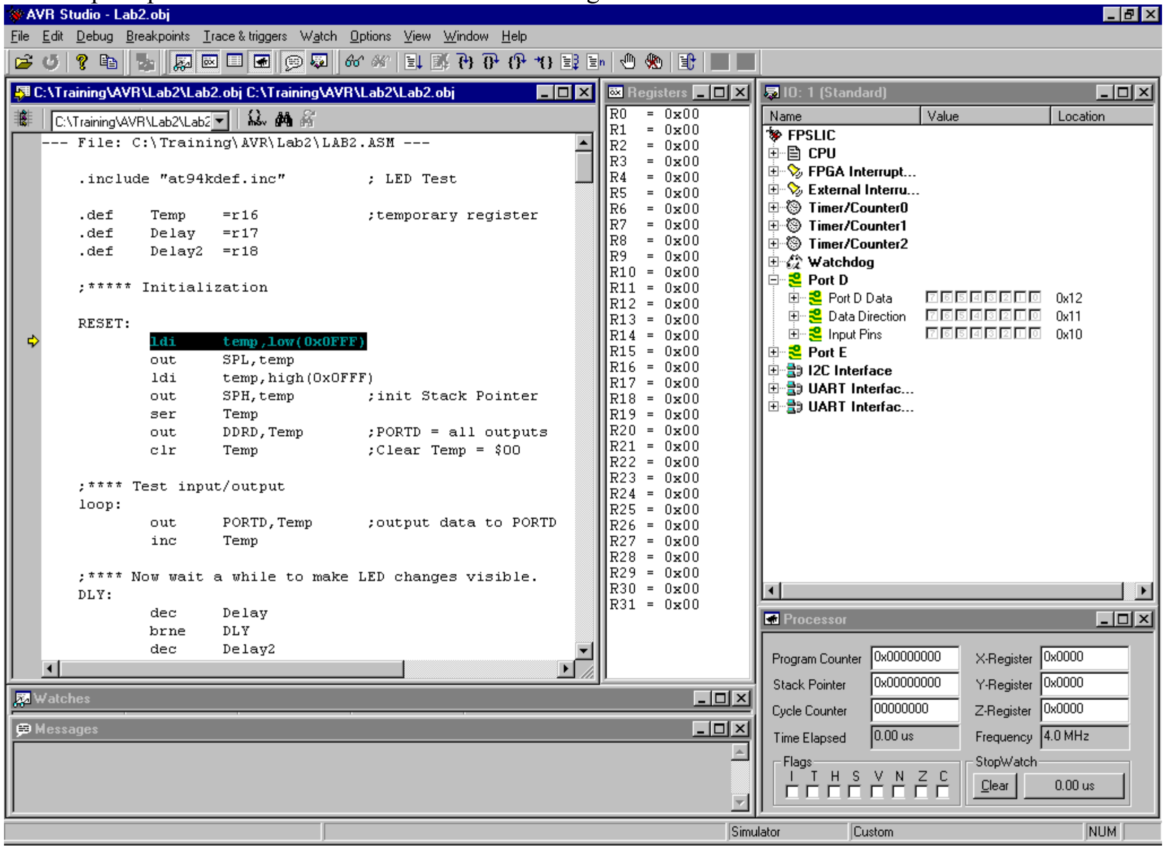
LAB 2: AVR Studio

AVR Studio

AVR Studio enables the user to fully control execution of programs on the built-in *AVR* Instruction Set Simulator. ***AVR Studio*** supports source level execution of Assembly programs assembled with the built in *AVR* Assembler, compiled with IAR's C Compiler or the ImageCraft C Compiler. In this lab ***AVR Studio*** will be used together with the built-in *AVR* Instruction Set Simulator.

AVR Studio Exercises

1. Open the ***AVR Studio*** program using the "SW Debugger" button within System Designer.
2. Select "Open" from the "File" menu in ***AVR Studio.*** Browse to the C:\TRAINING\AVR\LAB2\ folder and double-click on the LAB2.OBJ file. ***AVR Studio*** may prompt you for some device information, enter the following information if prompted. This will open the compiled source code.
    1. Device: `Custom`
    2. Program Memory: `16384`
    3. Data Memory: `4096`
    4. EEPROM: `0`
    5. I/O Size: `64`
    6. Frequency: `4 MHz`
3. Open the following additional views and arrange them as shown in **Figure 5**. This is done from the "View"menu. If you are prompted for the location of the .AIO file navigate to C:\SYSTEMDESIGNER\BIN\FPSLIC.AIO.


AVR Studio - Lab2.obj
File Edit Debug Breakpoints Trace & triggers Watch Options View Window Help
C:\Training\AVR\Lab2\Lab2.obj C:\Training\AVR\Lab2\Lab2.obj
C:\Training\AVR\Lab2\Lab2
Registers
IO: 1 (Standard)
Processor
Watches
Messages
Simulator
Custom
NUM

```
--- File: C:\Training\AVR\Lab2\LAB2.ASM ---

    .include "at94kdef.inc"         ; LED Test

    .def    Temp    =r16            ;temporary register
    .def    Delay   =r17
    .def    Delay2  =r18

    ;***** Initialization

    RESET:
            ldi     temp,low(0x0FFF)
            out     SPL,temp
            ldi     temp,high(0x0FFF)
            out     SPH,temp        ;init Stack Pointer
            ser     Temp
            out     DDRD,Temp       ;PORTD = all outputs
            clr     Temp            ;Clear Temp = $00

    ;**** Test input/output
    loop:
            out     PORTD,Temp      ;output data to PORTD
            inc     Temp

    ;**** Now wait a while to make LED changes visible.
    DLY:
            dec     Delay
            brne    DLY
            dec     Delay2
```

```
R0  = 0x00
R1  = 0x00
R2  = 0x00
R3  = 0x00
R4  = 0x00
R5  = 0x00
R6  = 0x00
R7  = 0x00
R8  = 0x00
R9  = 0x00
R10 = 0x00
R11 = 0x00
R12 = 0x00
R13 = 0x00
R14 = 0x00
R15 = 0x00
R16 = 0x00
R17 = 0x00
R18 = 0x00
R19 = 0x00
R20 = 0x00
R21 = 0x00
R22 = 0x00
R23 = 0x00
R24 = 0x00
R25 = 0x00
R26 = 0x00
R27 = 0x00
R28 = 0x00
R29 = 0x00
R30 = 0x00
R31 = 0x00
```

Name	Value	Location
FPSLIC		
CPU		
FPGA Interrupt...		
External Interru...		
Timer/Counter0		
Timer/Counter1		
Timer/Counter2		
Watchdog		
Port D		
Port D Data	7 6 5 4 3 2 1 0	0x12
Data Direction	7 6 5 4 3 2 1 0	0x11
Input Pins	7 6 5 4 3 2 1 0	0x10
Port E		
I2C Interface		
UART Interfac...		
UART Interfac...		

Program Counter	0x00000000	X-Register	0x0000
Stack Pointer	0x00000000	Y-Register	0x0000
Cycle Counter	00000000	Z-Register	0x0000
Time Elapsed	0.00 us	Frequency	4.0 MHz

Flags: I T H S V N Z C

StopWatch: Clear 0.00 us

Figure 5: AVR Studio

AVR Studio Questions

These questions will step you through several features of ***AVR Studio***. This will make you more familiar with ***AVR Studio***, which will help you debug your programs in LAB3, LAB4, and LAB5.

AVR Studio Basics

1. After ***AVR Studio*** environment is loaded, use the "Debug → Step Over" (F10) or the shortcut icon to single step five instruction. What is the value of the `Temp` variable (R16)? __________.
2. Single step another instruction (F10). How is `PORTD` configured? Use **Figure 6** to show your answer.

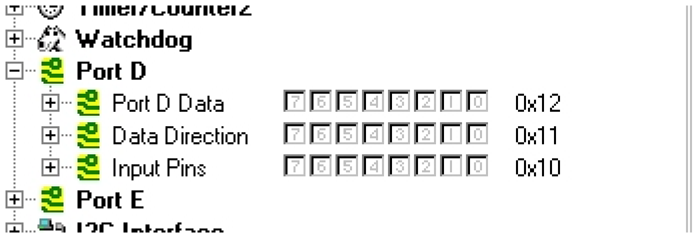


Figure 6: `PORTD`

3. Is `PORTD` configured as an "Input" or an "Output"? __________.

AVR Studio Advanced

1. Reset ***AVR Studio***. This is done by using the "Debug → Reset" (Shift + F5) from the menus.
2. Use the "View → New Memory View" (Alt + 4) to open a Memory Window. Once the Memory Window is open, select Program Memory and "16-Bits" **(Figure 7**). What are the contents at Program Memory location 0x0000? __________.

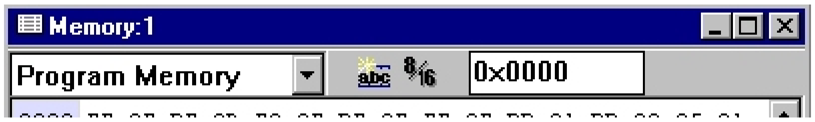


Figure 7: Memory Window Display Configuration

4. What are the Operational Codes (opcode) and instructions of `LEDTEST.OBJ` in Disassembly Mode? Toggle the Disassembly Mode icon to switch to Disassembly Mode. Complete **Table 1** to answer this question.

opcode	Instruction
EF0F	LDI R16,0xFF
EF0F	LDI R16,0xFF
BB01	OUT 0x11,R16
2700	CLR R16
BB02	OUT 0x12,R16
9503	INC R16
…	
F7E1	BRNE -0x04

Table 1: Disassembled `LEDTEST.OBJ`

4. Toggle the Disassembly Mode icon to switch back to Source Mode. Once this is complete, set a breakpoint at the "`inc Temp`" instruction. This is done by using "Breakpoints → Toggle Breakpoints" (F9). Use "Breakpoints → Show List" (Ctrl + B) to view the address of this breakpoint. What is the address? __________.

LAB 3: Using PORTD and PORTE

In this lab, a template, `SWITCH.ASM`, is provided to write a program to interface to the LEDs (LED1-LED8) and Switches (“SW5” and “SW6”) on the ATSTK94 FPSLIC Starter Kit using `PORTD` and `PORTE`. The function of this program is outlined below:

1. When “SW5” is pushed, the LEDs Count Up (Binary Counter Pattern).
2. When “SW6” is pushed, the LEDs Count Down (Binary Counter Pattern).
3. When no Switches are pushed, the LEDs are Off.

Design Overview-SWITCH.ASM

The `SWITCH.ASM` program is outlined in **Figure 8** and provides the basic template for this lab exercise. This file is located at C:\TRAINING\AVR\LAB3\.

```
;***** Switch Test

.include "at94kdef.inc"

.def Temp   =r16            ;temporary register
.def Test   =r19            ;temporary test register
.def Delay  =r17            ;Delay Variable
.def Delay2 =r18            ;Delay

;***** Initialization

RESET:
   ldi   temp,low(0x0FFF)
   out   SPL,Temp
   ldi   temp,high(0x0FFF)
   out   SPH,Temp           ;init Stack Pointer
   ser   Temp               ;Set Temp = $FF
   < AVR Instruction >      ;PORTD = all outputs
   < AVR Instruction >      ;Enable pull-ups on PORTE
   < AVR Instruction >      ;Set Temp = $00
   < AVR Instruction >      ;Initialize PORTD = $00


;**** Test SW5 and SW6

loop:
   sbis  PINE,0             ;Is SW5 pushed?
   dec   Temp               ;Decrement Temp
   < AVR Instruction >      ;Is SW6 pushed?
   < AVR Instruction >      ;Increment Temp
   < AVR Instruction >      ;Read PORTE
   < AVR Instruction >      ;Mask Off PE4 - PE7
   < AVR Instruction >      ;Check Switches
   < AVR Instruction >      ;Is SW5 or SW6 pushed?
   clr  Temp                ;SW5 & SW6 Off/LEDs Off
outled: out   PORTD,Temp    ;output data to PORTD

;**** LED changes visible with delay

DLY:
   Dec   Delay
   brne  DLY
   dec   Delay2
   brne  DLY
   rjmp  loop               ;repeat loop endlessly
```

Figure 8: `SWITCH.ASM`

LAB 3 Exercises

1. Use the template, `SWITCH.ASM`, and insert actual *AVR* Instructions to complete the program in **Figure 18**.
2. Once you have completed this program, assemble it using ***WAVRASM***, and debug any errors that are present.
3. Run the program using ***AVR Studio*** and the ATSTK94 FPSLIC Starter Kit, to verify correct operation.

Note:

1. You can use any of the AVR Software Tools to help debug your program.

LAB3 Hints and Tips

Please refer to the *"Programmable Logic and System Level ICs, FPSLIC Datasheet, 2000*" for more information on configuring `PORTD` and `PORTE`. To configure a `PORTD` as an Output, use the `DDRD` to select the direction. An LED is illuminated when the corresponding pin is high or "1". When using `PORTE` to interface the switches, the internal pull-ups are used to force the input pins to a known state when they are not pushed.

LAB 4: Using Timer/Counter0

The goal of LAB 4 is to become more familiar with `Timer/Counter0`. In this LAB, Timer/Counter0 will be configured to count based on “CLK/1024” and will be displayed to the LEDs when SW5 is pressed. The function of the program is as follows:

1. When “SW5” is pushed, the current count (binary counter pattern) will be displayed on LED1-LED8.
2. When “SW5” is not pushed, the current count will not be displayed on the LEDs (the LEDs will hold the last valid count after SW5 was released.)

Design Overview-TC0.ASM

The `TC0.ASM` program is outlined in **Figure 9** and provides the basic template for this lab exercise.
This file is located at C:\TRAINING\AVR\LAB4\.

```
;***** Using Timer/Counter0

.include "at94kdef.inc"

.def    Temp    =r16            ;temporary register
.def    Delay   =r17            ;Delay Variable
.def    Delay2  =r18            ;Delay

;***** Initialization

RESET:
        ldi     temp,low(0x0FFF)
        out     SPL,temp
        ldi     temp,high(0x0FFF)
        out     SPH,temp        ;init Stack Pointer
        <AVR Instruction>       ;Set Temp = $FF
        <AVR Instruction>       ;PORTD = all outputs
        <AVR Instruction>       ;Enable pull-ups on PORTE
        <AVR Instruction>       ;Clear Temp = $00
        <AVR Instruction>       ;Initialize PORTD = $00
        <AVR Instruction>       ;T/CO Initialization Value (CLK/1024)
        <AVR Instruction>       ;Load T/C0 Control Register

loop:
        <AVR Instruction>       ;Poll PORTE0
        <AVR Instruction>       ;Wait for Switch
        <AVR Instruction>       ;Get T/CO Count
        <AVR Instruction>       ;Output data to PORTD

DLY:    dec     Delay
        brne    DLY
        dec     Delay2
        brne    DLY
        rjmp    loop            ;repeat loop endlessly
```

Figure 9: `TC0.ASM`

LAB 4 Exercises

1. Use the template, `TC0.ASM`, and insert actual *AVR* Instructions to complete the program in **Figure 19.**
2. Once you have completed this program, assemble it and debug any errors that are present.
3. Use ***AVR Studio*** and ATSTK94 to verify correct operation.

LAB 4 Hints and Tips

Please refer to the *“Programmable Logic and System Level ICs, FPSLIC Datasheet, 2000*” for more information on configuring `Timer/Counter0`. “SW5” is mapped to an input pin on `PORTE`. When configuring `PORTE,` the pin should be configured as an input with an internal pull-up. `PORTD` is configured the same as in LAB 3.

LAB 5: Using the SRAM Memory

There are two goals for LAB 5: the first goal is become familiar with "Macros," while the second goal is to write a program to create a message in the SRAM. A template, `SRAM.ASM` **(Figure 20**) is provided as a starting point for this LAB. The function of this program is as follows:

1. Clear the first 64 (0 to $3F) locations of SRAM using the `SRAMCLEAR` Macro.
2. Write a message in SRAM using an `SRAMWRITE` Macro. The contents of this message are indicated in the `SRAM.ASM` template. *Note that a single ASCII Character is written to memory each time that the `SRAMWRITE` Macro is executed.*

Design Overview-SRAM.ASM

The `SRAM.ASM` program is outlined in **Figure 20** and provides the basic template for this lab exercise. This file is located at C:\TRAINING\AVR\LAB5\.

LAB 5 Exercises

1. Use the template, `SRAM.ASM`, and insert actual *AVR* Instructions to complete the program in **Figure 20.**
2. Once you have completed this program, assemble it using ***WAVRASM***, and debug any errors that are present.
3. To read the contents of SRAM, you can do the following:
    - Simulate the `SRAM.OBJ` program is ***AVR Studio*** and use "View → New Memory View" to view the contents of the SRAM.

LAB 5 Hints and Tips

- When using hex numbers in macros use the $VALUE notation, not the 0xVALUE notation.

LAB5 Questions

1. What is the message in SRAM? ________________________________.

```
;***** Using the SRAM Memory

.include "at94kdef.inc"
.def    Temp    =r16            ;temporary register
.def    Index   =r17            ;SRAM Address Index

.macro  SRAMWRITE
        ldi     Temp, @1        ;Get SRAM Data
        < AVR Instruction >     ;Store Data Direct at SRAM Location
.endmacro

.macro SRAMCLEAR
        ldi     XL,low(0x60)
        ldi     XH,high(0x60)   ;Initialize X Pointer = $0060
        < AVR Instruction >     ;Clear Index = $00
        < AVR Instruction >     ;Clear Temp = $00
loop :  < AVR Instruction >     ;Store Data and Post-increment X
        cpi     Index, @0       ;Compare Address with SRAM Size
        < AVR Instruction >     ;Address = Size, then done
        < AVR Instruction >     ;Inc Address to next location
        rjmp    loop
done:
.endmacro

RESET:  ldi     temp,low(0xFFF)
        out     SPL,temp
        ldi     temp,high(0xFFF)
        out     SPH,temp        ;init Stack Pointer
        ser     Temp            ;Set Temp = $FF
        out     DDRD,Temp       ;PORTB -> Output
        clr     Temp            ;Clear Temp = $00
        out     PORTD, Temp     ;Initialize PORTD
        < AVR Macro >           ;Clear EEPROM
        < AVR Macro >           ;SRAM ->Addr=$00, Data=$43
        < AVR Macro >           ;SRAM ->Addr=$01, Data=$4F
        < AVR Macro >           ;SRAM ->Addr=$02, Data=$4E
        < AVR Macro >           ;SRAM ->Addr=$03, Data=$47
        < AVR Macro >           ;SRAM ->Addr=$04, Data=$52
        < AVR Macro >           ;SRAM ->Addr=$05, Data=$41
        < AVR Macro >           ;SRAM ->Addr=$06, Data=$54
        < AVR Macro >           ;SRAM ->Addr=$07, Data=$55
        < AVR Macro >           ;SRAM ->Addr=$08, Data=$4C
        < AVR Macro >           ;SRAM ->Addr=$09, Data=$41
        < AVR Macro >           ;SRAM ->Addr=$0A, Data=$54
        < AVR Macro >           ;SRAM ->Addr=$0B, Data=$49
        < AVR Macro >           ;SRAM ->Addr=$0C, Data=$4F
        < AVR Macro >           ;SRAM ->Addr=$0D, Data=$4E
        < AVR Macro >           ;SRAM ->Addr=$0E, Data=$53
        < AVR Macro >           ;SRAM ->Addr=$0F, Data=$21
loop:   ldi     Temp, $81       ;Turn LED7 & LED0 ON
        out     PORTD,Temp      ;output data to PORTD
        rjmp    loop
```

Figure 10: `SRAM.ASM`

LAB 6: Reaction Tester

LAB6 combines LAB1 to LAB5 in a real application. The application is a reaction tester that measures how long it takes from an LED being lit to a button being pressed. The final code should be run on the ATSTK94 FPSLIC Starter Kit Evaluation Board.
To ease the task of generating the program a file named `START1.ASM` is provided. This file includes some ready-made functions, and provides the skeleton program. The code writing should follow the following steps:

Step 1. Generate Random Number

Use the Linear congruential method:

- $X_{n+1} = (19X_n + 27) \bmod 255$
- X_n is the old number.
- Use the AVR Hardware Multiplier for multiplication.
- Use r0 as the old value.

Step 2. Variable Delay

Generate a variable delay of length X*0.01s

- Use timer Counter 1.
- Count 40,000 Cycles at 4 MHz.
- Enable Clear Timer on Compare Match and count FCK/1 (TCCR1B)
  Write 40,000 into OCR1A register
- Decrement X each time the OCF1A flag is set. Exit when X is 0.
- Turn on an LED when delay has expired.

Let X be the pseudo-random number from step 1.

Step 3. Measure Reaction Time

Measure time from LED lit until key pressed

- Use same delay in step 2, but modify slightly.
  - This time, exit when any key is pressed.
  - Time used is then -X
- Display time on LEDs in Binary.
- Let LEDs be lit for 2.5 seconds, then turn off.

Step 4. High Score

Store the best time in the SRAM

- Compare each time against the high-score
- If time is better, write the new high-score
- Blink result if it is high-score. Else turn LEDs on static.

Step 5. Reduce Power Consumption

Use Power-down Mode when waiting for player in Main.

- Enable power down mode in MCUR. Enable External Interrupt 0 and 1 in EIMF. Execute the sleep instruction to enter power down mode.

Use Idle Mode in Delay routine

- Enable Timer/Counter1 Compare Match interrupt (TIMSK). Enable idle mode in MCUR.

Solution LAB3

```
;***** Switch Test

.include "at94kdef.inc"

.def   Temp    =r16          ;temporary register
.def   Test    =r19          ;temporary test register
.def   Delay   =r17          ;Delay Variable
.def   Delay2  =r18          ;Delay

;***** Initialization

RESET:
       ldi    temp,low(0x0FFF)
       out    SPL,temp
       ldi    temp,high(0x0FFF)
       out    SPH,temp       ;init Stack Pointer
       ser    Temp           ;Set Temp = $FF
       out    DDRD, Temp     ;PORTD = all outputs
       out    PORTE, Temp    ;Enable pull-ups on PORTE
       clr    Temp           ;Set Temp = $00
       out    PORTD, Temp    ;Initialize PORTD = $00

;**** Test SW0 and SW1

loop:  sbis   PINE,0         ;Is SW5 pushed?
       dec    Temp           ;Decrement Temp
       sbis   PINE,1         ;Is SW6 pushed?
       inc    Temp           ;Increment Temp
       in     Test, PINE     ;Read PORTE
       andi   Test, 0x0F     ;Mask Off PE4 – PE7
       cpi    Test, 0x00     ;Check Switches
       brne   outled         ;Is SW5 or SW6 pushed?
       clr    Temp           ;SW5 & SW6 Off/LEDs Off
outled: out   PORTD,Temp     ;output data to PORTD

;**** LED changes visible with delay

DLY:   dec    Delay
       brne   DLY
       dec    Delay2
       brne   DLY
       rjmp   loop           ;repeat loop endlessly
```

Solution LAB4

```
;***** Using Timer/Counter0

.include "at94kdef.inc"

.def    Temp    =r16           ;temporary register
.def    Delay   =r17           ;Delay Variable
.def    Delay2  =r18           ;Delay

;***** Initialization

RESET:
        ldi     temp,low(0x0FFF)
        out     SPL,temp
        ldi     temp,high(0x0FFF)
        out     SPH,temp        ;init Stack Pointer
        ser     temp            ;Set Temp = $FF
        out     DDRD, temp      ;PORTD = all outputs
        out     PORTE, temp     ;Enable pull-ups on PORTE
        clr     temp            ;Clear Temp = $00
        out     PORTD, temp     ;Initialize PORTD = $00
        ldi     temp, 0x05      ;T/CO Initialization Value (CLK/1024)
        out     TCCR0, temp     ;Load T/C0 Control Register

loop:   sbis    PINE, 0         ;Poll PORTE0
        rjmp    loop            ;Wait for Switch
        in      Temp, TCNT0     ;Get T/CO Count
        out     PORTD, Temp     ;Output data to PORTD


DLY:    dec     Delay
        brne    DLY
        dec     Delay2
        brne    DLY
        rjmp    loop            ;repeat loop endlessly
```

Solution LAB5

```
;***** Using the SRAM Memory

.include "at94kdef.inc"
.def    Temp    =r16           ;temporary register
.def    Index   =r17           ;SRAM Address Index

.macro SRAMCLEAR
       ldi     XL,low(0x60)
       ldi     XH,high(0x60)   ;Initialize X Pointer = $0060
       clr     Index          ;Clear Index = $00
       clr     Temp           ;Clear Temp = $00
loop :  st      X+, Temp        ;Store Data and Increment X
       cpi     Index, @0       ;Compare Address with SRAM Size
       breq    done           ;Address = Size, then done
       inc     Index          ;Inc Address to next location
       rjmp    loop
done:
.endmacro


.macro  SRAMWRITE
       ldi     Temp, @1        ;Get SRAM Data
       sts     @0, Temp        ;Store Data Direct at SRAM Location
.endmacro

RESET:  ldi     temp,low(0xFFF)
       out     SPL,temp
       ldi     temp,high(0xFFF)
       out     SPH,temp        ;init Stack Pointer
       ser     Temp           ;Set Temp = $FF
       out     DDRD,Temp       ;PORTD -> Output
       clr     Temp           ;Clear Temp = $00
       out     PORTD, Temp     ;Initialize PORTD
       SRAMCLEAR $3F          ;Clear EEPROM
       SRAMWRITE $60,$43       ;SRAM ->Addr=$00, Data=$43
       SRAMWRITE $61,$4F       ;SRAM ->Addr=$01, Data=$4F
       SRAMWRITE $62,$4E       ;SRAM ->Addr=$02, Data=$4E
       SRAMWRITE $63,$47       ;SRAM ->Addr=$03, Data=$47
       SRAMWRITE $64,$52       ;SRAM ->Addr=$04, Data=$52
       SRAMWRITE $65,$41       ;SRAM ->Addr=$05, Data=$41
       SRAMWRITE $66,$54       ;SRAM ->Addr=$06, Data=$54
       SRAMWRITE $67,$55       ;SRAM ->Addr=$07, Data=$55
       SRAMWRITE $68,$4C       ;SRAM ->Addr=$08, Data=$4C
       SRAMWRITE $69,$41       ;SRAM ->Addr=$09, Data=$41
       SRAMWRITE $6A,$54       ;SRAM ->Addr=$0A, Data=$54
       SRAMWRITE $6B,$49       ;SRAM ->Addr=$0B, Data=$49
       SRAMWRITE $6C,$4F       ;SRAM ->Addr=$0C, Data=$4F
       SRAMWRITE $6D,$4E       ;SRAM ->Addr=$0D, Data=$4E
       SRAMWRITE $6E,$53       ;SRAM ->Addr=$0E, Data=$53
       SRAMWRITE $6F,$21       ;SRAM ->Addr=$0F, Data=$21
loop:   ldi     Temp, $81       ;Turn LED7 & LED0 ON
       out     PORTD,Temp      ;output data to PORTD
       rjmp    loop
```