FPSLIC Training Series (AVR based designs)

Software Requirements

- **System Designer**
  - **ImageCraft C Compiler AVR**
  - **Atmel AVR Studio**
- **CPS**

Hardware Requirements

- **ATSTK94 Starter Kit**

LAB 1 – ImageCraft C Compiler AVR and Programming Utility

LAB 2 - AVR Studio

LAB 3 - Using PORTD and PORTE

LAB 4 - Using Timer/Counter0

LAB 5 - Using the SRAM Memory

LAB 6 - Reaction Tester

LAB 1: ImageCraft C Compiler AVR and Programming Utility

The goal of LAB1 is to familiarize yourself with the ImageCraft C Compiler *AVR* (30-day evaluation version included with System Designer), the *FPSLIC* Programming Utility (***CPS***), and the ATSTK94 FPSLIC Starter Kit. This LAB will teach you how to compile and download a program to the Starter Kit using the Atmel *FPSLIC* and ImageCraft Tools.

Design Overview-LEDTEST.C

The `LEDTEST.C` program **(Figure** 1) is a C 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 <ioat94k.h>


/* initAT94K - Device Initialization */
void initAT94K(void)
{
        DDRD = 0xFF;                                // PORTD = All Outputs
}


/* Main Program  - Test Input/Output */
void main(void)
{
        unsigned char delay = 0;
        unsigned char delay2 = 0;
        unsigned char temp = 0;

        initAT94K();                                // Initialize Device

        for(;;)                                     // Repeat Loop Endlessly
        {
                PORTD = temp++;                     // Output Data to PORTD

                /* Now Wait a While - Make LED Changes Visible */
                do
                {
                        do
                        { } while (--delay != 0);

                } while (--delay2 != 0);
        }
}
```

Figure 1: `LEDTEST.C`

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".

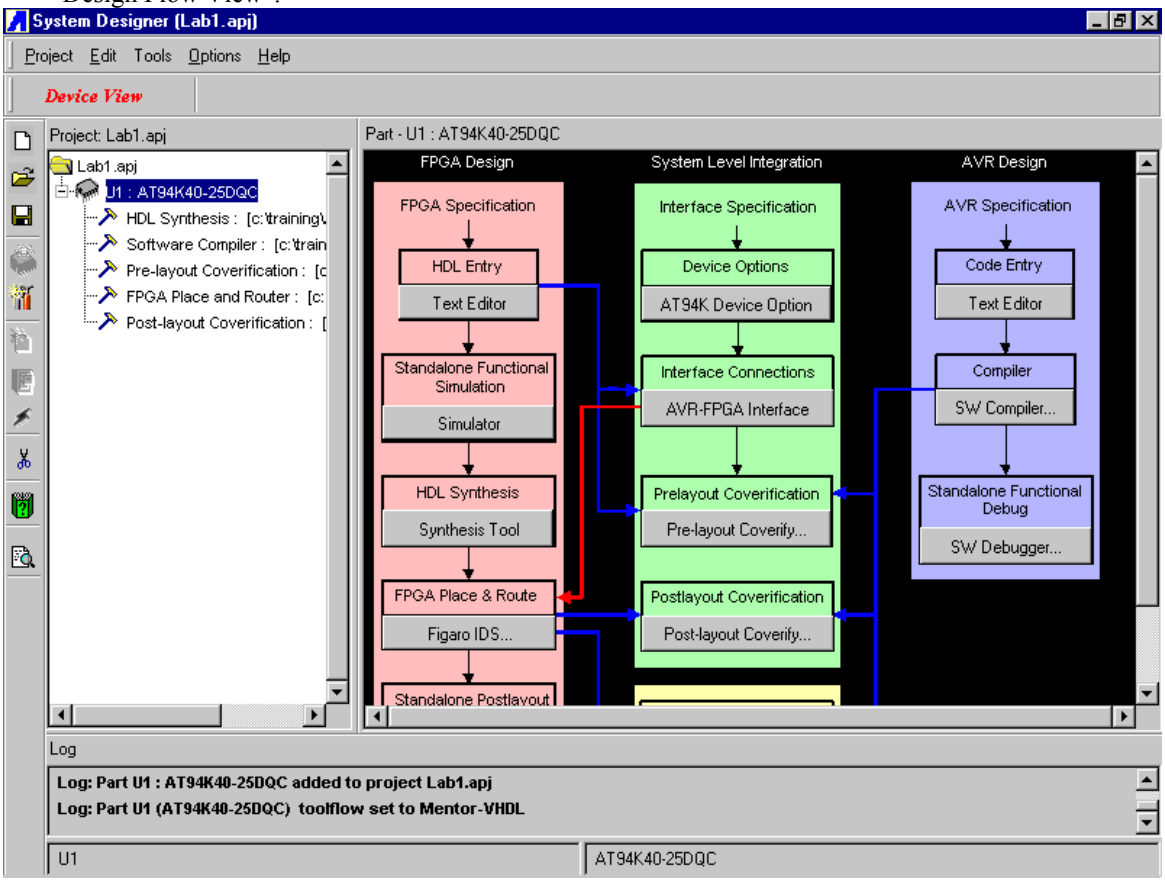

Figure 1: System Designer Design Flow

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 Stand-alone) 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 the ***ImageCraft C Compiler AVR*** for source code entry, we will choose the second option.

Writing C in ImageCraft C Compiler AVR

The ImageCraft C Compiler AVR translates C 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 C Compiler generates fixed code allocations; consequently, no linking is necessary.

ImageCraft C Compiler AVR Exercises

1. Open ImageCraft C Compiler AVR by using either the Shortcut or the Start Menu.

If you would like to make the ImageCraft C Compiler your default software compiler, that is associate the "SW Compiler" button with the ImageCraft C Compiler, follow the steps outlined below.

- Right Click on the "SW Compiler" button and select "Change Tool Settings"
- Navigate to the C:\ICC\BIN\ and select ICCAVRIDE.EXE, press "OK"
- System Designer then prompts for an update of the Plug-In, if "Yes" is pressed ImageCraft will become the default Software Compiler for all projects, if "No" is selected ImageCraft will only be active for the current project.

2. Create a new project `LEDTEST` by choosing Project -> New and navigate to the C:\TRAINING\AVR\LAB1 directory.
3. Configure the new project by choosing Project -> Options, from the "Target" screen select "AT94K 16K Program" for "Device Configuration" (See **Figure 2**).

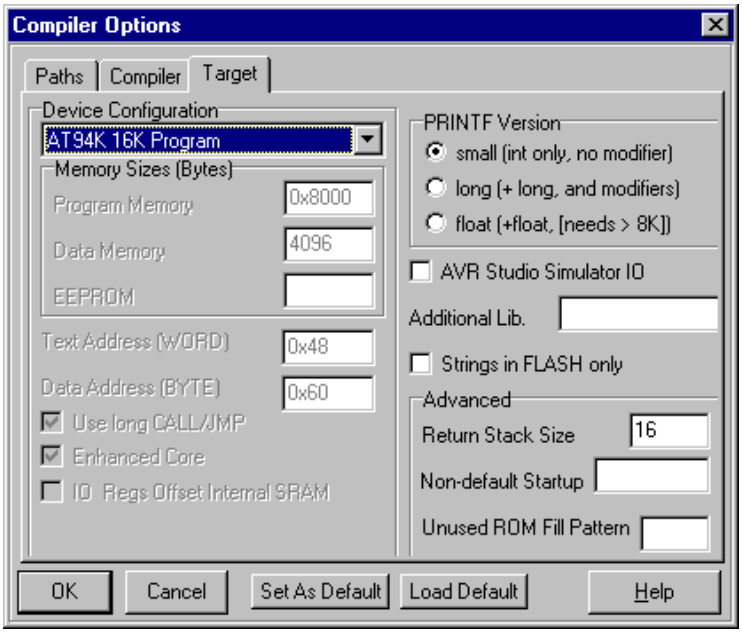

Figure 2: *ImageCraft C Compiler* Options – Target Tab

4. Press "OK"
5. Add the `LEDTEST.C` file to the current project by choosing Project -> Add File(s)… and select `LEDTEST.C`. You should now see `LEDTEST.C` as a member of the current project in the Project Window. You may also double-click on `LEDTEST.C` from the project window to open the file for editing.
6. Build and Compile the project by selecting "Make Project" from the "Project" menu.

ImageCraft C Compiler Questions

1. What percentage of the device does the `LEDTEST` program use? Answer: _____

Programming the Device

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

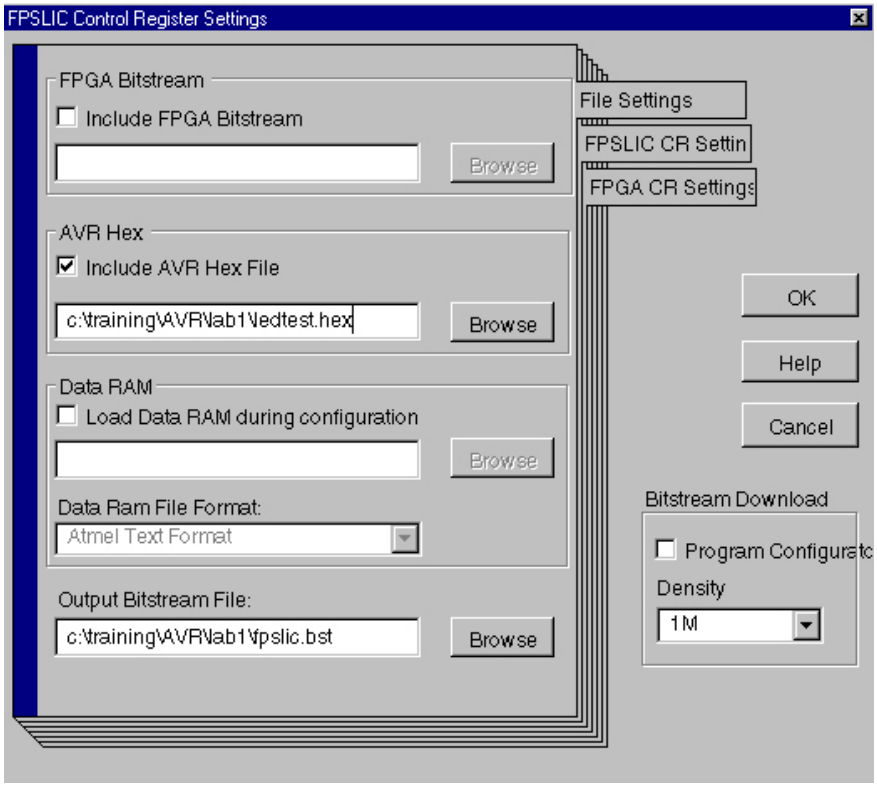

Figure 3: Bitstream File Settings Window

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**.

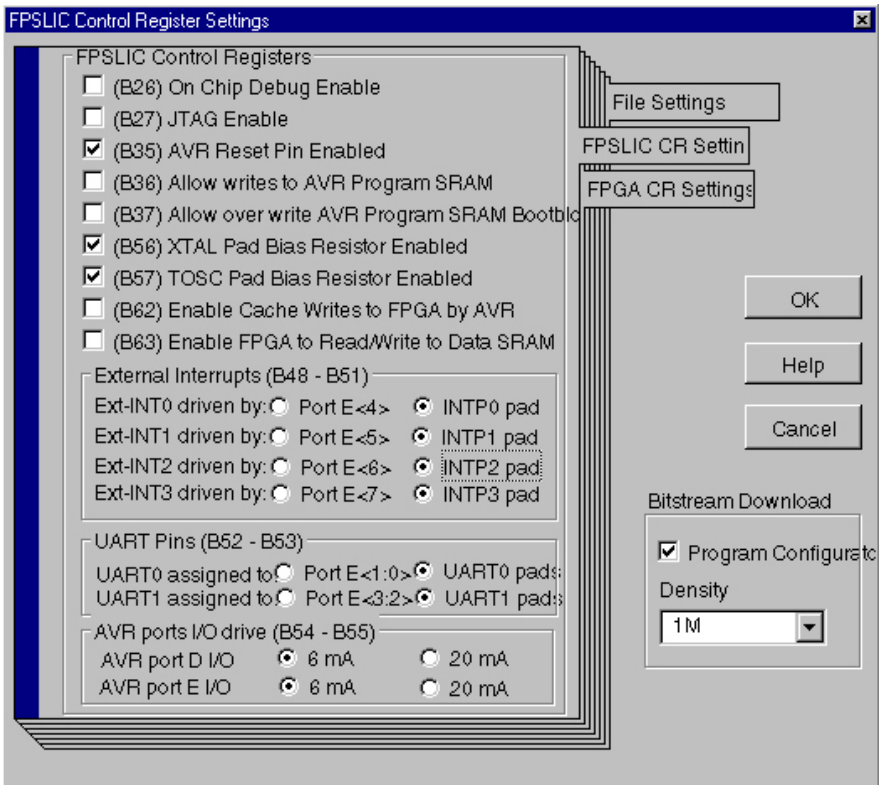

Figure 4: Bitstream Control Register Settings

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.COF file. ***AVR Studio*** may prompt you for some device information, enter the following information if prompted. This will open the compiled source code.
   - Device: `Custom`
   - Program Memory: `16384`
   - Data Memory: `4096`
   - EEPROM: `0`
   - I/O Size: `64`
   - 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.

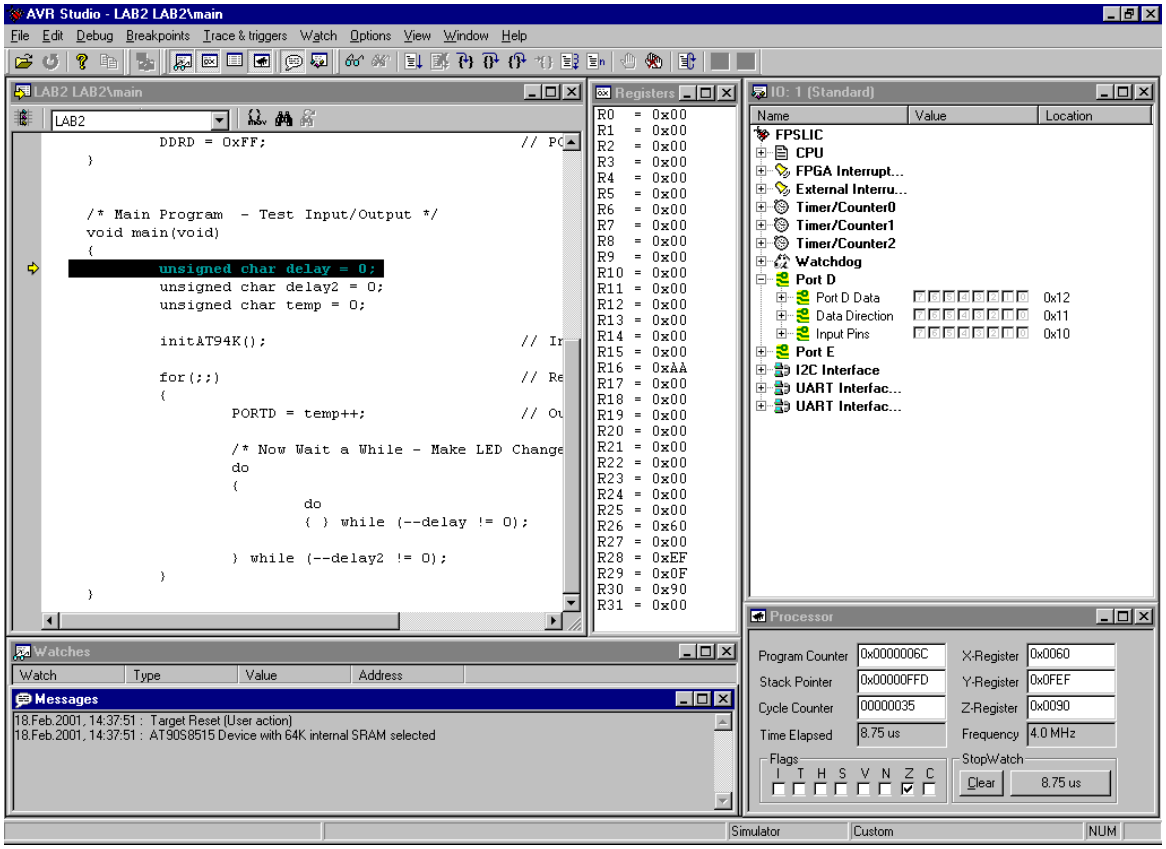

Figure 5: AVR Studio

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 (R24)? _____.
2. How is `PORTD` configured? Use **Figure 6** to show your answer.

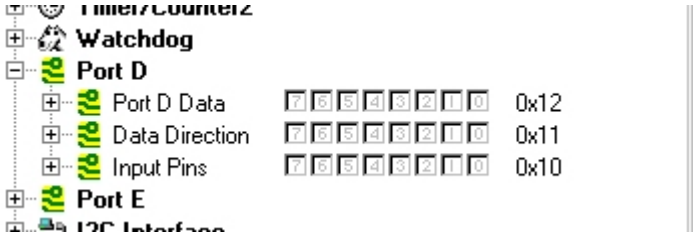

Figure 6: `PORTD`

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? _____.

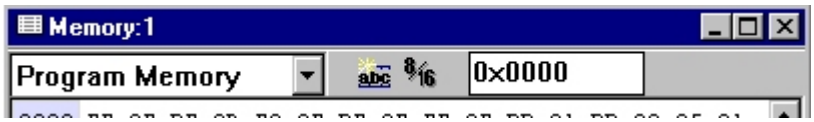

Figure 7: Memory Window Display Configuration

3. What are the Operational Codes (opcode) and instructions of `LEDTEST.C` in Disassembly Mode? Toggle the Disassembly Mode icon to switch to Disassembly Mode. Complete **Table 1** to answer this question, start with the `main` function.

opcode	Instruction
24AA	CLR R10
DFF9	RCALL initAT94K
BA22	OUT 0x12, R2
…	
9508	RET
FFFF	SBRS R31, 7

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 "`PORTD = 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.C, 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.C

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

```
#include <ioat94k.h>

/* initAT94K - Device Initialization */
void initAT94K(void)
{
        <AVR Instruction>                       // Initialize PORTD = 0x00
        <AVR Instruction>                       // PORTD = All Outputs
        <AVR Instruction>                       // Enable Pull-Ups on PORTE
}


/* Main Program  - Test Input/Output */
void main(void)
{
        unsigned char delay = 0;
        unsigned char delay2 = 0;
        unsigned char temp = 0;

        initAT94K();                            // Initialize Device

        for(;;)                                 // Repeat Loop Endlessly
        {
                if ( PINE & (1 << PE0) ) {      // Is SW5 (PE0) Pushed?
                        temp--;                 // Decrement temp
                }

                if ( <AVR Instruction> ) {      // Is SW6 (PE1) Pushed?
                        <AVR Instruction>       // Increment temp
                }

                if ( <AVR Instruction> ) {      // Is SW5 or SW6 Pushed & Mask Off PE4 - PE7
                        <AVR Instruction>       // SW5 & SW6 Off => LEDs Off
                }

                PORTD = temp;                   // Output Data to PORTD

                /* Now Wait a While - Make LED Changes Visible */
                do
                {
                        do
                        { } while (--delay != 0);

                } while (--delay2 != 0);
        }
}
```

Figure 8: SWITCH.C

LAB 3 Exercises

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

Notes:

1. *You can use any of the AVR Software Tools to help debug your program.*
2. *The instructions for the AVR are located in the "AVR RISC Microcontroller Databook, 1999."*

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.C

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

```
#include <ioat94k.h>

/* initAT94K - Device Initialization */
void initAT94K(void)
{
        <AVR Instruction>                       // PORTD = All Outputs
        <AVR Instruction>                       // Initialize PORTD = 0x00
        <AVR Instruction>                       // Enable Pull-Ups on PORTE
        <AVR Instruction>                       // Initialize T/C0 as CLK/1024
}


/* Main Program */
void main(void)
{
        unsigned char delay = 0;
        unsigned char delay2 = 0;
        unsigned char temp = 0;

        initAT94K();                            // Initialize Device

        for(;;)                                 // Repeat Loop Endlessly
        {
                while ( <AVR Instruction> );    // Poll PE0 - Wait for Switch

                <AVR Instruction>               // Output T/C0 on PORTD

                /* Now Wait a While - Make LED Changes Visible */
                do
                {
                        do
                        { } while (--delay != 0);

                } while (--delay2 != 0);
        }
}
```

Figure 9: TC0.C

LAB 4 Exercises

1. Use the template, TC0.C, 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.C` **(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` Function.
2. Write a message in SRAM using an `SRAMWRITE` Function. The contents of this message are indicated in the `SRAM.C` template. *Note that a single ASCII Character is written to memory each time that the `SRAMWRITE` Macro is executed.*

Design Overview-SRAM.C

The `SRAM.C` program is outlined in **Figure 10** 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.C`, and insert actual *AVR* Instructions to complete the program in **Figure 20.**
2. Once you have completed this program, assemble it using ***ImageCraft C Compiler AVR***, and debug any errors that are present.
3. To read the contents of SRAM, you can do the following:
    - Simulate the `SRAM.COF` 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 0xVALUE notation.

LAB5 Questions

1. What is the message in SRAM? _____.

```
#include <ioat94k.h>


/* Currently, ImageCraft v6.20 (January 2, 2001) the only way to locate
 * variables in specific SRAM locations is to use In-Line Assembly and
 * change the .org to the desired SRAM Location.
 */
asm(".area memory(abs)\n"
    ".org 0x0060\n"
    "_locatedSRAM :: .blkb 64");

extern unsigned char locatedSRAM[64];           // Array of Located SRAM


/* SRAMClear - Clear SRAM */
void SRAMClear(unsigned char * arrayptr, unsigned char size)
{
        unsigned char index = size;

        do
        {
                <AVR Instruction>               // Start SRAM Write
        } while ( <AVR Instruction> );          // SRAM Access Complete?
}


/* SRAMWrite - Write Data to SRAM */
void SRAMWrite(unsigned char * arrayptr, unsigned char address, unsigned char data)
{
        <AVR Instruction>                       // Start SRAM Write
}


/* initAT94K - Device Initialization */
void initAT94K(void)
{
        DDRD = 0xFF;                            // PORTD = All Outputs
        PORTD = 0x00;                           // Initialize PORTD = 0x00

        <AVR Function>                          // Clear SRAM
        <AVR Function>                          // SRAM -> Addr = 0x00, Data = 0x43
        <AVR Function>                          // SRAM -> Addr = 0x01, Data = 0x4F
        <AVR Function>                          // SRAM -> Addr = 0x02, Data = 0x4E
        <AVR Function>                          // SRAM -> Addr = 0x03, Data = 0x47
        <AVR Function>                          // SRAM -> Addr = 0x04, Data = 0x52
        <AVR Function>                          // SRAM -> Addr = 0x05, Data = 0x41
        <AVR Function>                          // SRAM -> Addr = 0x06, Data = 0x54
        <AVR Function>                          // SRAM -> Addr = 0x07, Data = 0x55
        <AVR Function>                          // SRAM -> Addr = 0x08, Data = 0x4C
        <AVR Function>                          // SRAM -> Addr = 0x09, Data = 0x41
        <AVR Function>                          // SRAM -> Addr = 0x0A, Data = 0x54
        <AVR Function>                          // SRAM -> Addr = 0x0B, Data = 0x49
        <AVR Function>                          // SRAM -> Addr = 0x0C, Data = 0x4F
        <AVR Function>                          // SRAM -> Addr = 0x0D, Data = 0x4E
        <AVR Function>                          // SRAM -> Addr = 0x0E, Data = 0x53
        <AVR Function>                          // SRAM -> Addr = 0x0F, Data = 0x21
}


/* Main Program  - Test Input/Output */
void main(void)
{
        unsigned char temp = 0;

        initAT94K();                            // Initialize Device

        for(;;)                                 // Repeat Loop Endlessly
        {
                PORTD = 0x81;                   // Turn LED1 & LED8 ON
        }
}
```

Figure 10: SRAM.C

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

```
#include <ioat94k.h>


/* initAT94K - Device Initialization */
void initAT94K(void)
{
      PORTD = 0x00;                    // Initialize PORTD = 0x00
      DDRD = 0xFF;                     // PORTD = All Outputs
      PORTE = 0xFF;                    // Enable Pull-Ups on PORTE
}


/* Main Program  - Test Input/Output */
void main(void)
{
      unsigned char delay = 0;
      unsigned char delay2 = 0;
      unsigned char temp = 0;

      initAT94K();                     // Initialize Device

      for(;;)                          // Repeat Loop Endlessly
      {
            if ( PINE & (1 << PE0) ) {     // Is SW5 (PE0) Pushed?
                  temp--;                  // Decrement temp
            }

            if ( PINE & (1 << PE1) ) {     // Is SW6 (PE1) Pushed?
                  temp++;                  // Increment temp
            }

            if ( (PINE  & 0x0F) == 0 ) {   // Is SW5 or SW6 Pushed?
                  temp = 0x00;             // SW5 & SW6 Off => LEDs Off
            }

            PORTD = temp;                  // Output Data to PORTD

            /* Now Wait a While - Make LED Changes Visible */
            do
            {
                  do
                  { } while (--delay != 0);

            } while (--delay2 != 0);
      }
}
```

Solution LAB4

```
#include <ioat94k.h>


/* initAT94K - Device Initialization */
void initAT94K(void)
{
      DDRD = 0xFF;                    // PORTD = All Outputs
      PORTD = 0x00;                   // Initialize PORTD = 0x00
      PORTE = 0xFF;                   // Enable Pull-Ups on PORTE
      TCCR0 = 0x05;                   // Initialize T/C0 as CLK/1024
}


/* Main Program */
void main(void)
{
      unsigned char delay = 0;
      unsigned char delay2 = 0;
      unsigned char temp = 0;

      initAT94K();                    // Initialize Device

      for(;;)                         // Repeat Loop Endlessly
      {
            while (!(PINE & (1 << PE0)));   // Poll PE0 - Wait for Switch

            PORTD = TCNT0;            // Output T/C0 on PORTD

            /* Now Wait a While - Make LED Changes Visible */
            do
            {
                  do
                  { } while (--delay != 0);

            } while (--delay2 != 0);
      }
}
```

Solution LAB5

```
#include <ioat94k.h>

/* Currently, ImageCraft v6.20 (January 2, 2001) the only way to locate
 * variables in specific SRAM locations is to use In-Line Assembly and
 * change the .org to the desired SRAM Location.
 */
asm(".area memory(abs)\n"
   ".org 0x0060\n"
   "_locatedSRAM :: .blkb 64");

extern unsigned char locatedSRAM[64];          // Array of Located SRAM

/* SRAMClear - Clear SRAM */
void SRAMClear(unsigned char * arrayptr, unsigned char size)
{
      unsigned char index = size;

      do
      {
            arrayptr[index] = 0;          // Start SRAM Write
      } while (--index != 0);             // SRAM Access Complete?
}


/* SRAMWrite - Write Data to SRAM */
void SRAMWrite(unsigned char * arrayptr, unsigned char address, unsigned char data)
{
      arrayptr[address] = data;             // Start SRAM Write
}


/* initAT94K - Device Initialization */
void initAT94K(void)
{
      DDRD = 0xFF;                    // PORTD = All Outputs
      PORTD = 0x00;                   // Initialize PORTD = 0x00

      SRAMClear(&locatedSRAM[0], 64);        // Clear SRAM
      SRAMWrite(&locatedSRAM[0], 0, 0x43);   // SRAM -> Addr = 0x00, Data = 0x43
      SRAMWrite(&locatedSRAM[0], 1, 0x4F);   // SRAM -> Addr = 0x01, Data = 0x4F
      SRAMWrite(&locatedSRAM[0], 2, 0x4E);   // SRAM -> Addr = 0x02, Data = 0x4E
      SRAMWrite(&locatedSRAM[0], 3, 0x47);   // SRAM -> Addr = 0x03, Data = 0x47
      SRAMWrite(&locatedSRAM[0], 4, 0x52);   // SRAM -> Addr = 0x04, Data = 0x52
      SRAMWrite(&locatedSRAM[0], 5, 0x41);   // SRAM -> Addr = 0x05, Data = 0x41
      SRAMWrite(&locatedSRAM[0], 6, 0x54);   // SRAM -> Addr = 0x06, Data = 0x54
      SRAMWrite(&locatedSRAM[0], 7, 0x55);   // SRAM -> Addr = 0x07, Data = 0x55
      SRAMWrite(&locatedSRAM[0], 8, 0x4C);   // SRAM -> Addr = 0x08, Data = 0x4C
      SRAMWrite(&locatedSRAM[0], 9, 0x41);   // SRAM -> Addr = 0x09, Data = 0x41
      SRAMWrite(&locatedSRAM[0], 10, 0x54);  // SRAM -> Addr = 0x0A, Data = 0x54
      SRAMWrite(&locatedSRAM[0], 11, 0x49);  // SRAM -> Addr = 0x0B, Data = 0x49
      SRAMWrite(&locatedSRAM[0], 12, 0x4F);  // SRAM -> Addr = 0x0C, Data = 0x4F
      SRAMWrite(&locatedSRAM[0], 13, 0x4E);  // SRAM -> Addr = 0x0D, Data = 0x4E
      SRAMWrite(&locatedSRAM[0], 14, 0x53);  // SRAM -> Addr = 0x0E, Data = 0x53
      SRAMWrite(&locatedSRAM[0], 15, 0x21);  // SRAM -> Addr = 0x0F, Data = 0x21
}
/* Main Program  - Test Input/Output */
void main(void)
{
      unsigned char temp = 0;

      initAT94K();                    // Initialize Device

      for(;;)                         // Repeat Loop Endlessly
      {
            PORTD = 0x81;             // Turn LED1 & LED8 ON
      }
}
```

Solution LAB6

```
#include <ioat94k.h>
#include <macros.h>

/* Currently, ImageCraft v6.20 (January 2, 2001) the only way to locate
 * variables in specific SRAM locations is to use In-Line Assembly and
 * change the .org to the desired SRAM Location.
 */
asm(".area memory(abs)\n"
   ".org 0x0060\n"
   "_locatedSRAM :: .blkb 64");

extern unsigned char locatedSRAM[64];          // Array of Located SRAM

static unsigned long x0   = 65536;
static unsigned char time =     0;

#pragma interrupt_handler      INT0_handler:EXT_INT0
#pragma interrupt_handler      INT1_handler:EXT_INT1
#pragma interrupt_handler      OC1A_handler:TIM1_COMPA


/* ISR - External Interrupt 0 */
void INT0_handler(void) { }                 // Used for Wake-Up Only


/* ISR - External Interrupt 1 */
void INT1_handler(void) { }                 // Used for Wake-Up Only


/* ISR - T/C1 Output Compare A */
void OC1A_handler(void)
{
      unsigned char backupSREG = 0;

      backupSREG = SREG;
      time++;
      SREG = backupSREG;
}


/* initAT94K - Device Initialization */
void initAT94K(void)
{
      unsigned char index = 0;

      do
      {
            locatedSRAM[index] = 0xFF;      // Set SRAM = $FF
      } while (--index < 64);

      PORTD = 0x00;                       // Initialize PORTD = 0x00
      DDRD = 0xFF;                        // PORTD = All Outputs
      PORTE = 0xFF;                       // Enable Pull-Ups on PORTE

      EIMF = 0x33;                        // Enable External Interrupt 0 & 1
}


/* random - Generates Random Number */
/* Using the linear congruential method to generate a pseudorandom number.  We
 * chose four integers: the Modulus m, multiplier a, increment c and seed x0,
 * with 2 <= a < m, 0 <= c < m, and 0 <= x0 < m.  We then generate the sequence
 * {Xn} with 0 <= Xn < m for all n by using the congruence {Xn+1} = (aXn + c) mod m
 *
 * In this example we have selected a = 19, c = 27, m = 255 and x0 = 65536.
 */
unsigned char random(void)
```

```
{
	unsigned char a  = 19;
	unsigned char c  = 27;
	unsigned char m  = 255;

	unsigned char result = 0;

	x0 = (((a * x0) + c) % m);
	result = (char) x0;

	return (result);
}


/* delay - Waits delay time * ~100 ms */
void delay(unsigned char delayTime)
{
	TCCR1B = 0x00;
	TCNT1H = 0x00;
	TCNT1L = 0x00;                    // Clear and Stop T/C1

	OCR1A = (400000/64);              // 400000 Cycles gives .1 sec delay

	TCCR1B = (1 << CTC1) + 3;         // Clear T/C1 on Compare Match and count
                                  // FClk/64
	time = 0;
	TIMSK |= (1 << OCIE1A);
	MCUR |= (1 << SE);                // Enable Idle-Mode

	Do
	{
		asm("sleep");
	} while (time != delayTime);      // Continue until 0
}


/* cheating */
void cheating(void)
{
	PORTD = 0xAA;
}


/* getTime */
/* Gets time from LED lit until button pushed.  Measures time from 0.01 seconds
 * to 2.55 seconds.
 */
unsigned char getTime(void)
{
	time   = 0;
	TCNT1  = 0;
	TCCR1B = 1;                       // Count CLK/64
	OCR1A  = 32767;

	TIMSK |= (1 << OCIE1A);
	MCUR  |= (1 << SE);               // Enable Idle-Mode

	for(;;)
	{
		if (++time == 0)
		{
			PORTD = 0xA5;
			Return(--time);       // Exit if time to late (time = 255)
		}

		time--;

		if ((PINE & 0x0F) != 0)       // Exit if key pressed
			return(time);
	}
}
```

```
/* checkIfBest */
/* Function to test if current time is the best
 * Returns with T-Flag Set if it is
 */
void checkIfBest(unsigned char reactionTime)
{
	if (reactionTime < locatedSRAM[0])
	{
		locatedSRAM[0] = reactionTime;
		asm("set");
	}
	else asm("clt");
}


/* displayTime */
/* Shows the Score
 * Blinks the Score if it is a High Score
 */
void displayTime(unsigned char reactionTime)
{
	unsigned char displayCount = 10;

	do
	{
		PORTD = reactionTime;
		delay(5);

		if ( SREG & (1 << 6) )
			PORTD = 0x00;

		delay(5);
	} while (--displayCount != 0);
}


/* Main Program */
void main(void)
{
	unsigned char reactionTime = 0;

	initAT94K();

	for(;;)
	{
		MCUR  = ( (1 << SE) | (1 << SM1) );	// Enable Power-Down Mode
		SEI();				// Enable Global Interrupts

		asm("sleep");			// Wait Until a player wants to
					// play
		delay(random());			// Wait for Time-Out

		delay(10);			// Wait 1 Second

		if ( (PINE & 0x0F) > 0)		// Read Switches SW5 - SW8
			cheating();		// Cheating if button pushed
					// before LED
		PORTD = (1 << PD0);			// Light Up LED

		reactionTime = getTime();
		checkIfBest(reactionTime);
		displayTime(reactionTime);

		PORTD = 0x00;			// Turn Off LEDs to save Power
	}
}
```