Reaction Time Game Project

Team Names:

Youssef Taha Saad Taha

Ziad Ahmed Mohamed Abdelfattah

Moamen Mohamed Ahmed Hassan

Supervised by:

Dr. Lamiaa El-Refaey

Eng. Mahmoud Nawar

Table of Contents:

1 - Introduction

The Reaction Time Game is an interactive project designed to measure the user's reaction time. Built around the TM4C123GH6PM microcontroller, the system challenges users to press a button corresponding to an LED that lights up. The challenge is that the user needs to press the button during an interval of time and if the button isn't pressed during this interval of time, then the user fails. The game is enhanced with levels of difficulty as the levels get higher and the user will have less time to press the button before the LEDs turn off. The user is also provided with real-time feedback about the number of attempts he has left on a Nokia 5110 LCD.

2- System Design

2.1 Hardware Components:

- TM4C123GH6PM Microcontroller
- LEDs (Red, Green)
- Push-button switches
- Potentiometer
- Resistors (330 Ω for LEDs, pull-up resistors for switches)
- Nokia 5110 LCD
- UART communication interface

2.2 Software Components:

- Embedded C programming
- Keil uVision IDE for code development
- TivaWare library for peripheral control

3- Software Implementation

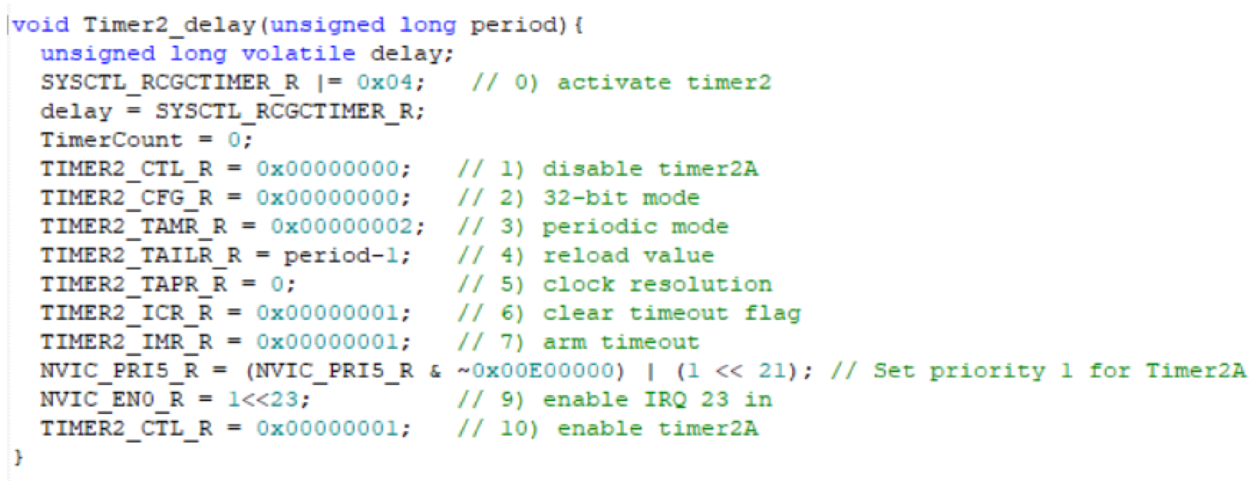
As shown previously, the project uses Timer2A, UART, Nokia5110, GPIOs, LED, and ADC.

3.1. Timer2A:

This timer function is used in coordination with its handler function to randomize the lighting up of the LEDs. The handler function is also used to show the user the number of attempts he has left before he runs out.

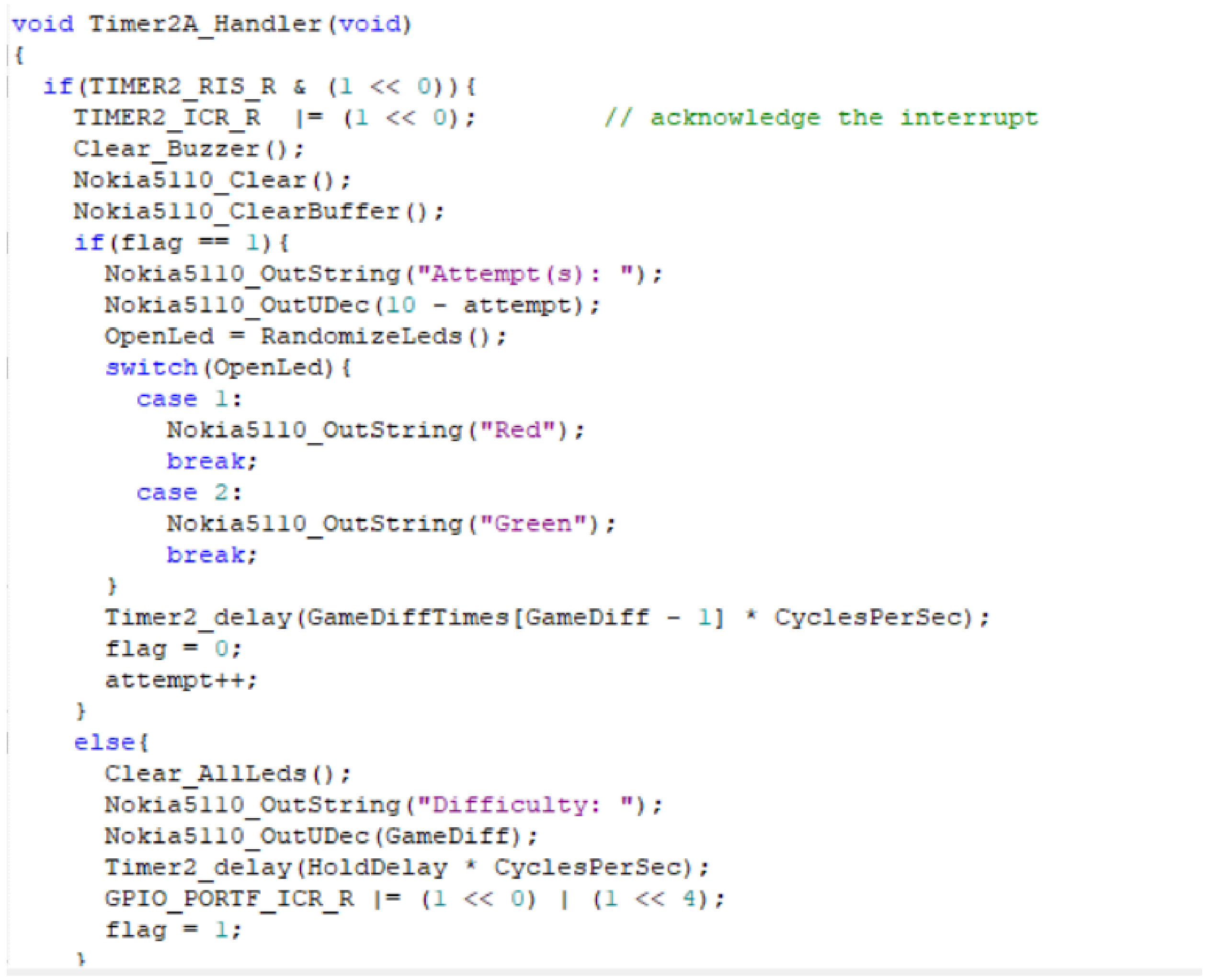
Timer2_delay function:

This function is used to initialize Timer2A with a value written as its parameter. After the value inside the parameter is zero the interrupt handler will be called. The timer function can only have two possible values which are: the time value which is set by the user choosing a level of difficulty, and the time value which is set in between every attempt.



```
void Timer2_delay(unsigned long period){
  unsigned long volatile delay;
  SYSCTL_RCGCTIMER_R |= 0x04;   // 0) activate timer2
  delay = SYSCTL_RCGCTIMER_R;
  TimerCount = 0;
  TIMER2_CTL_R = 0x00000000;    // 1) disable timer2A
  TIMER2_CFG_R = 0x00000000;    // 2) 32-bit mode
  TIMER2_TAMR_R = 0x00000002;   // 3) periodic mode
  TIMER2_TAILR_R = period-1;    // 4) reload value
  TIMER2_TAPR_R = 0;            // 5) clock resolution
  TIMER2_ICR_R = 0x00000001;    // 6) clear timeout flag
  TIMER2_IMR_R = 0x00000001;    // 7) arm timeout
  NVIC_PRI5_R = (NVIC_PRI5_R & ~0x00E00000) | (1 << 21); // Set priority 1 for Timer2A
  NVIC_EN0_R = 1<<23;           // 9) enable IRQ 23 in
  TIMER2_CTL_R = 0x00000001;    // 10) enable timer2A
}
```

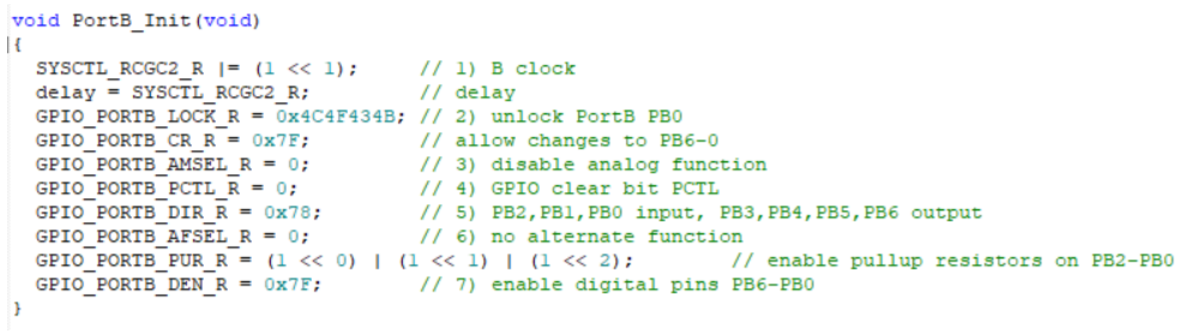
Timer2A_Handler function:


```
void Timer2A_Handler(void)
{
  if(TIMER2_RIS_R & (1 << 0)){
    TIMER2_ICR_R  |= (1 << 0);        // acknowledge the interrupt
    Clear_Buzzer();
    Nokia5110_Clear();
    Nokia5110_ClearBuffer();
    if(flag == 1){
      Nokia5110_OutString("Attempt(s): ");
      Nokia5110_OutUDec(10 - attempt);
      OpenLed = RandomizeLeds();
      switch(OpenLed){
        case 1:
          Nokia5110_OutString("Red");
          break;
        case 2:
          Nokia5110_OutString("Green");
          break;
      }
      Timer2_delay(GameDiffTimes[GameDiff - 1] * CyclesPerSec);
      flag = 0;
      attempt++;
    }
    else{
      Clear_AllLeds();
      Nokia5110_OutString("Difficulty: ");
      Nokia5110_OutUDec(GameDiff);
      Timer2_delay(HoldDelay * CyclesPerSec);
      GPIO_PORTF_ICR_R |= (1 << 0) | (1 << 4);
      flag = 1;
    }
```

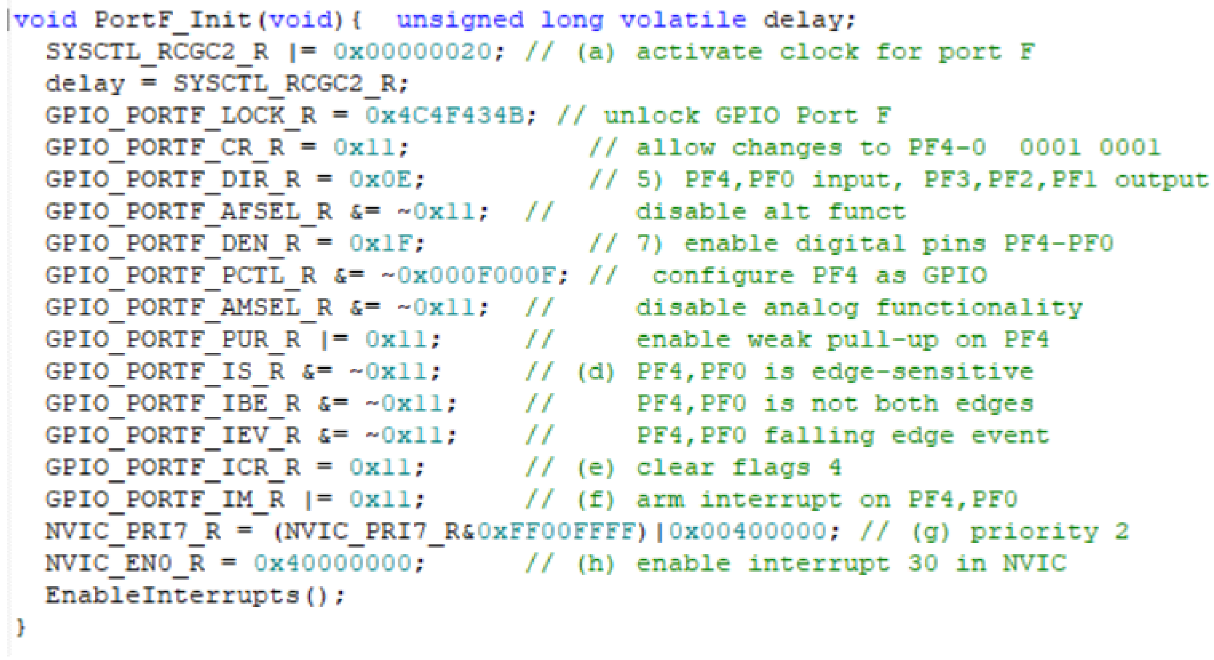
First, the handler function clears the screen of the Nokia5110. Then, it shows the number of attempts the user has left. Every user is allowed to attempt 10 attempts. The handler checks if the user is in delay time between the attempts; if so, it calls the RandomizeLeds function which is used to light up an LED in front of the user at random. Other than the user being in the time between the attempts, the handler will give the user a Be Ready message which means that an LED is going to light up.

3.2. GPIO_PortB:

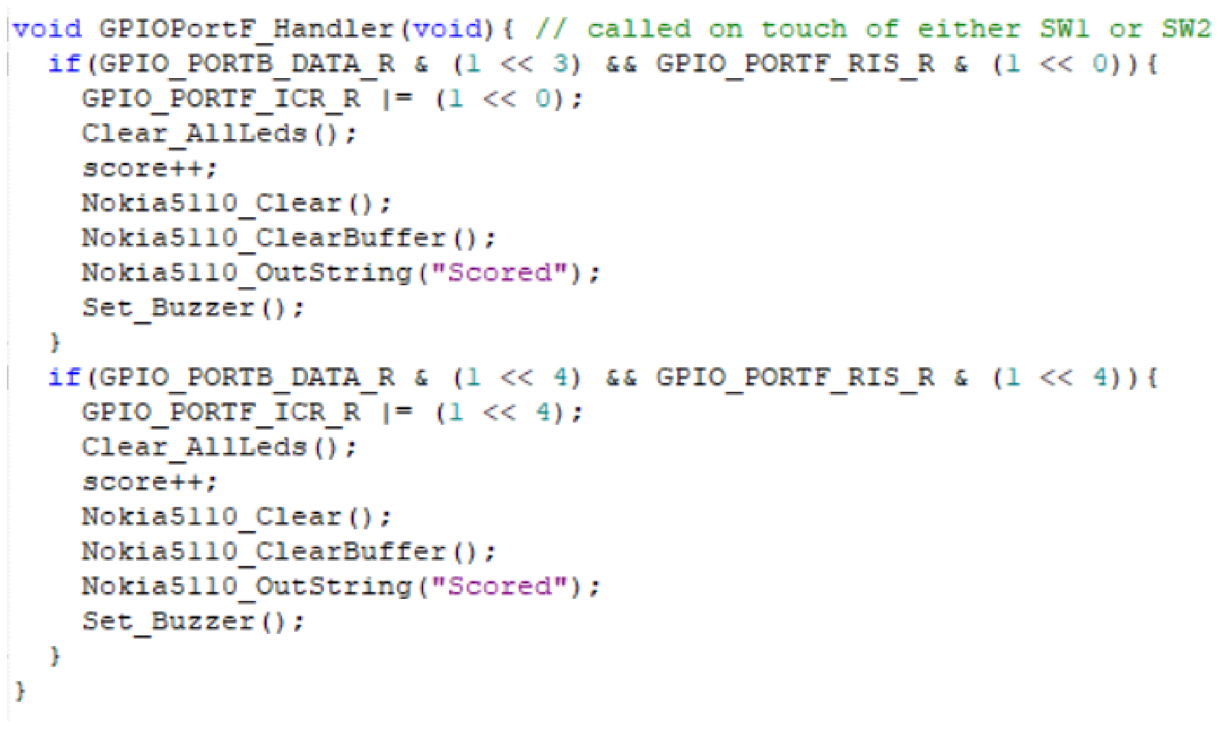
The GPIO_PortB is used completely for the input and the output of the game as it is responsible for lighting the LEDs, setting the buzzer when there is a wrong attempt, and handling the interrupts with the handler function on the switches.



```c
void PortB_Init(void)
{
  SYSCTL_RCGC2_R |= (1 << 1);     // 1) B clock
  delay = SYSCTL_RCGC2_R;         // delay
  GPIO_PORTB_LOCK_R = 0x4C4F434B; // 2) unlock PortB PB0
  GPIO_PORTB_CR_R = 0x7F;         // allow changes to PB6-0
  GPIO_PORTB_AMSEL_R = 0;         // 3) disable analog function
  GPIO_PORTB_PCTL_R = 0;          // 4) GPIO clear bit PCTL
  GPIO_PORTB_DIR_R = 0x78;        // 5) PB2,PB1,PB0 input, PB3,PB4,PB5,PB6 output
  GPIO_PORTB_AFSEL_R = 0;         // 6) no alternate function
  GPIO_PORTB_PUR_R = (1 << 0) | (1 << 1) | (1 << 2);        // enable pullup resistors on PB2-PB0
  GPIO_PORTB_DEN_R = 0x7F;        // 7) enable digital pins PB6-PB0
}
```



```c
void PortF_Init(void){  unsigned long volatile delay;
  SYSCTL_RCGC2_R |= 0x00000020; // (a) activate clock for port F
  delay = SYSCTL_RCGC2_R;
  GPIO_PORTF_LOCK_R = 0x4C4F434B; // unlock GPIO Port F
  GPIO_PORTF_CR_R = 0x11;           // allow changes to PF4-0  0001 0001
  GPIO_PORTF_DIR_R = 0x0E;          // 5) PF4,PF0 input, PF3,PF2,PF1 output
  GPIO_PORTF_AFSEL_R &= ~0x11;  //     disable alt funct
  GPIO_PORTF_DEN_R = 0x1F;          // 7) enable digital pins PF4-PF0
  GPIO_PORTF_PCTL_R &= ~0x000F000F; //  configure PF4 as GPIO
  GPIO_PORTF_AMSEL_R &= ~0x11;  //     disable analog functionality
  GPIO_PORTF_PUR_R |= 0x11;     //     enable weak pull-up on PF4
  GPIO_PORTF_IS_R &= ~0x11;     // (d) PF4,PF0 is edge-sensitive
  GPIO_PORTF_IBE_R &= ~0x11;    //     PF4,PF0 is not both edges
  GPIO_PORTF_IEV_R &= ~0x11;    //     PF4,PF0 falling edge event
  GPIO_PORTF_ICR_R = 0x11;      // (e) clear flags 4
  GPIO_PORTF_IM_R |= 0x11;      // (f) arm interrupt on PF4,PF0
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF00FFFF)|0x00400000; // (g) priority 2
  NVIC_EN0_R = 0x40000000;      // (h) enable interrupt 30 in NVIC
  EnableInterrupts();
}
```



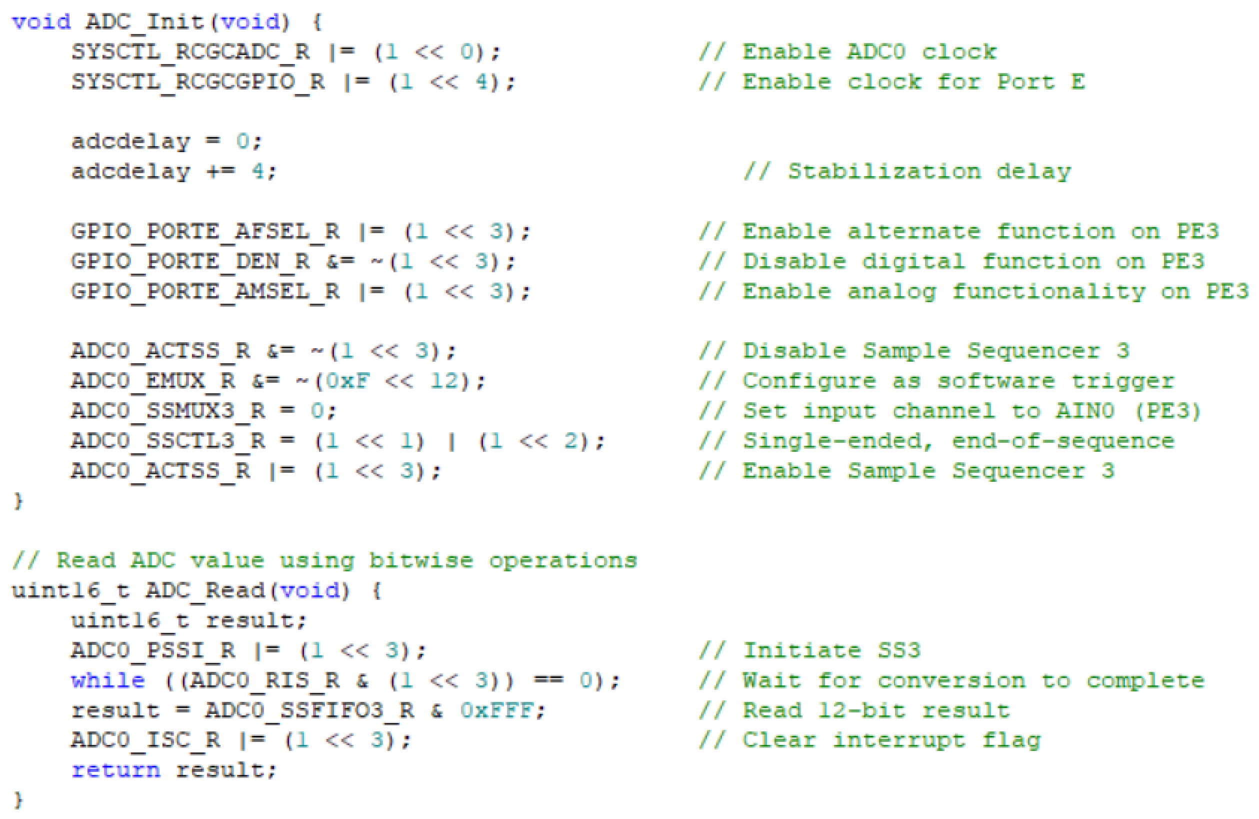
```
void GPIOPortF_Handler(void){ // called on touch of either SW1 or SW2
  if(GPIO_PORTB_DATA_R & (1 << 3) && GPIO_PORTF_RIS_R & (1 << 0)){
    GPIO_PORTF_ICR_R |= (1 << 0);
    Clear_AllLeds();
    score++;
    Nokia5110_Clear();
    Nokia5110_ClearBuffer();
    Nokia5110_OutString("Scored");
    Set_Buzzer();
  }
  if(GPIO_PORTB_DATA_R & (1 << 4) && GPIO_PORTF_RIS_R & (1 << 4)){
    GPIO_PORTF_ICR_R |= (1 << 4);
    Clear_AllLeds();
    score++;
    Nokia5110_Clear();
    Nokia5110_ClearBuffer();
    Nokia5110_OutString("Scored");
    Set_Buzzer();
  }
}
```

Also, the handler function checks for the pressed switch and its LED if the user pressed right or wrong there are two variables `attempt`, and `score`. The `attempt` variable is used to count the wrong and right attempts to make the game with only ten attempts per game and the `score` variable is used to count the right attempts only.

3.3. Analog-to-Digital converter:

The analog-to-digital converter is used for the potentiometer. The ADC uses the voltage on the potentiometer to allow the user to choose the desired difficulty.

ADC1_initialize:



```
void ADC_Init(void) {
    SYSCTL_RCGCADC_R |= (1 << 0);             // Enable ADC0 clock
    SYSCTL_RCGCGPIO_R |= (1 << 4);            // Enable clock for Port E

    adcdelay = 0;
    adcdelay += 4;                               // Stabilization delay

    GPIO_PORTE_AFSEL_R |= (1 << 3);           // Enable alternate function on PE3
    GPIO_PORTE_DEN_R &= ~(1 << 3);            // Disable digital function on PE3
    GPIO_PORTE_AMSEL_R |= (1 << 3);           // Enable analog functionality on PE3

    ADC0_ACTSS_R &= ~(1 << 3);                // Disable Sample Sequencer 3
    ADC0_EMUX_R &= ~(0xF << 12);              // Configure as software trigger
    ADC0_SSMUX3_R = 0;                        // Set input channel to AIN0 (PE3)
    ADC0_SSCTL3_R = (1 << 1) | (1 << 2);      // Single-ended, end-of-sequence
    ADC0_ACTSS_R |= (1 << 3);                 // Enable Sample Sequencer 3
}

// Read ADC value using bitwise operations
uint16_t ADC_Read(void) {
    uint16_t result;
    ADC0_PSSI_R |= (1 << 3);                  // Initiate SS3
    while ((ADC0_RIS_R & (1 << 3)) == 0);     // Wait for conversion to complete
    result = ADC0_SSFIFO3_R & 0xFFF;          // Read 12-bit result
    ADC0_ISC_R |= (1 << 3);                   // Clear interrupt flag
    return result;
}
```

ADC Initialization (ADC_Init)

- Enables the clock for ADC0 and Port E.
- Configures PE3 as an analog input pin (disables digital functions and enables analog mode).
- Sets up Sample Sequencer 3 to use software triggering and single-ended input (AIN0).
- Configures the ADC to generate interrupts upon completing a conversion and marks the end-of-sequence.

ADC Read (ADC_Read)

- Starts a conversion on Sample Sequencer 3.
- Waits for the conversion to complete by polling the ADC interrupt status.

- Reads the 12-bit result from the FIFO and clears the interrupt flag to prepare for the next conversion.

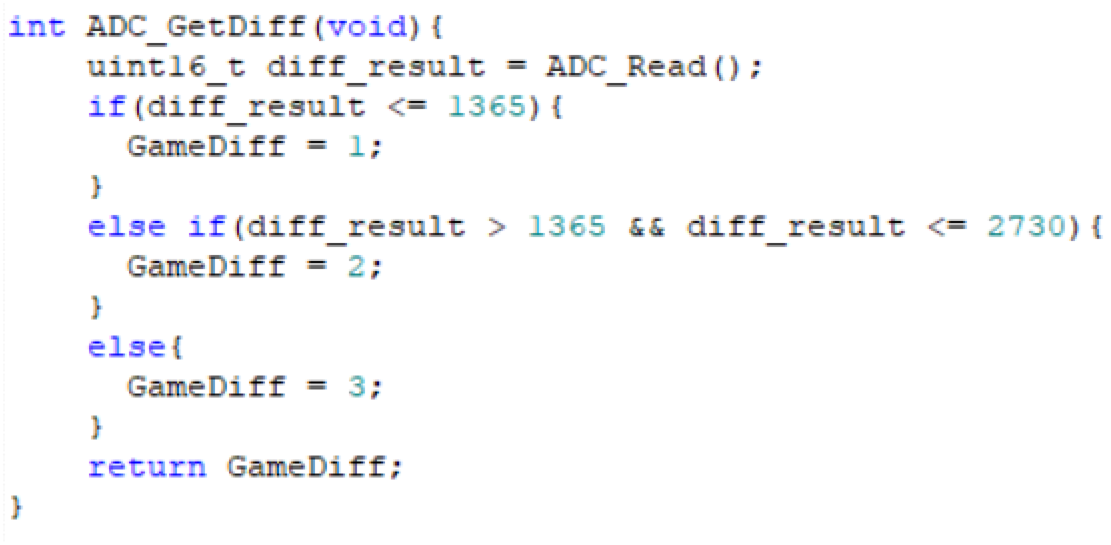
This setup lets you read analog values from a potentiometer or sensor connected to PE3.

ADC_GetDiff:

This function, ADC_GetDiff, is a method that chooses the difficulty of the game. The entire ADC range (0-4095) is divided into three equal sections:

- Level 1: 0 - 1365
- Level 2: 1366 - 2730
- Level 3: 2731 - 4095

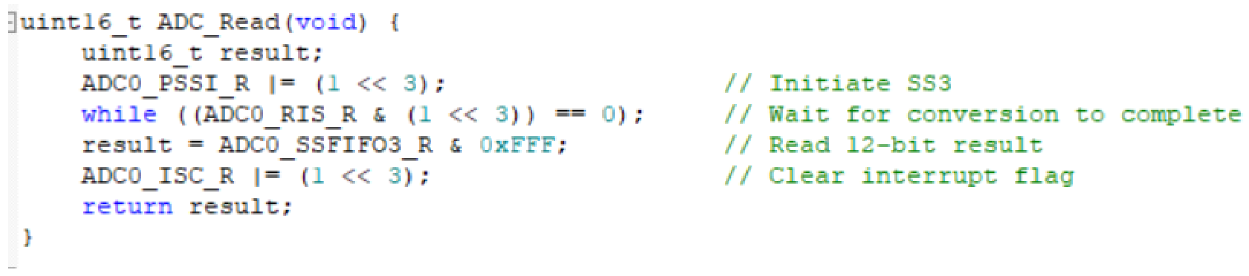
The corresponding difficulty levels of the game (GameDiff) are assigned based on the ADC value (diff_result).



```
int ADC_GetDiff(void){
    uint16_t diff_result = ADC_Read();
    if(diff_result <= 1365){
      GameDiff = 1;
    }
    else if(diff_result > 1365 && diff_result <= 2730){
      GameDiff = 2;
    }
    else{
      GameDiff = 3;
    }
    return GameDiff;
}
```

ADC_Read:

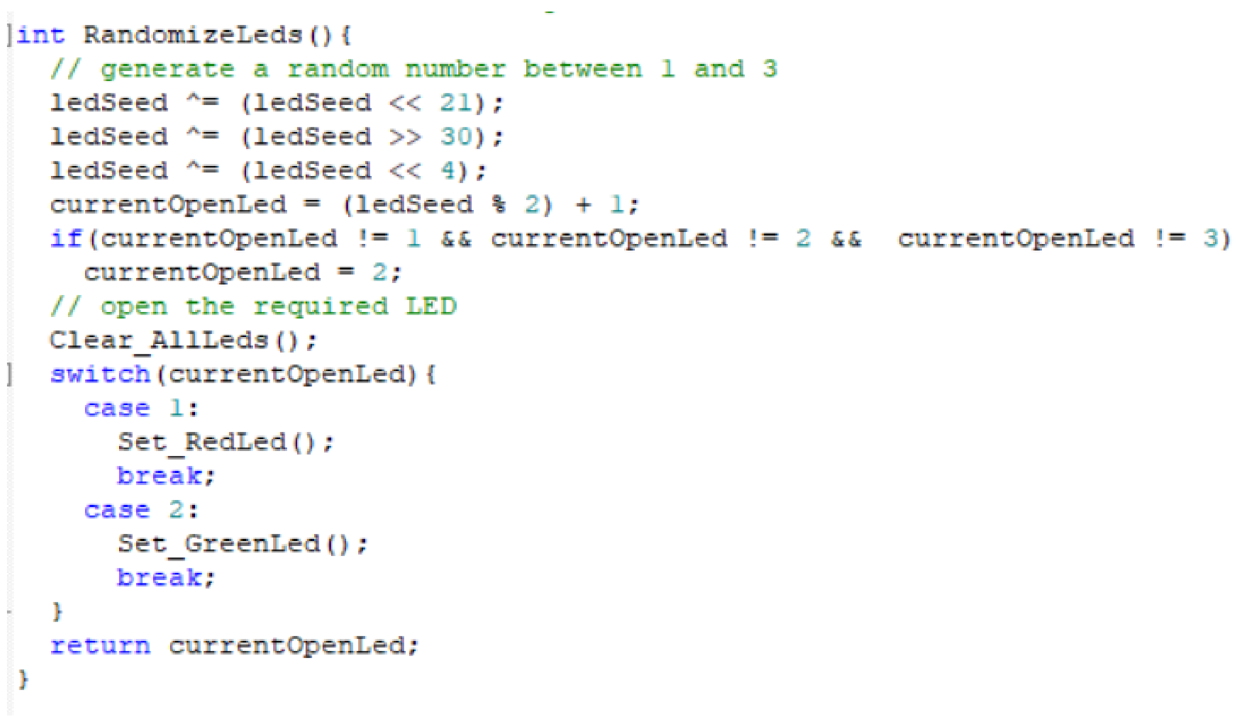
The `**ADC_Read**` function reads a 12-bit analog-to-digital conversion result from an ADC module. It begins by initiating a conversion on Sample Sequencer 3 (SS3) by setting the corresponding bit in the `**ADC0_PSSI_R**` register. The function then waits for the conversion to complete by monitoring the `**ADC0_RIS_R**` register until the appropriate bit is set, indicating readiness. Once complete, it retrieves the 12-bit result from the `**ADC0_SSFIFO3_R**` register, which stores the output of the conversion, and masks it with **0xFFF** to ensure only the 12 least significant bits are considered. Finally, it clears the interrupt flag for SS3 by writing to the `**ADC0_ISC_R**` register and returns the converted digital value as a 16-bit unsigned integer **(uint16_t)**.



```
uint16_t ADC_Read(void) {
    uint16_t result;
    ADC0_PSSI_R |= (1 << 3);                 // Initiate SS3
    while ((ADC0_RIS_R & (1 << 3)) == 0);    // Wait for conversion to complete
    result = ADC0_SSFIFO3_R & 0xFFF;         // Read 12-bit result
    ADC0_ISC_R |= (1 << 3);                  // Clear interrupt flag
    return result;
}
```

LED:

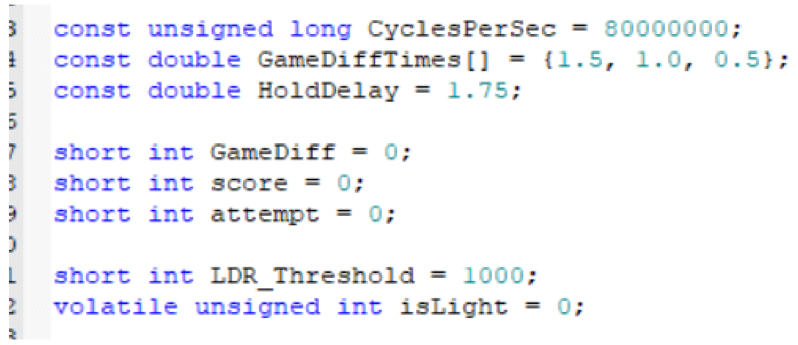
This file is used to initialize the LEDs in port B, Set the Buzzer, and Clear the LEDs. It also has a function called "RandomizeLeds" This function is used to set a random LED.



```
int RandomizeLeds(){
  // generate a random number between 1 and 3
  ledSeed ^= (ledSeed << 21);
  ledSeed ^= (ledSeed >> 30);
  ledSeed ^= (ledSeed << 4);
  currentOpenLed = (ledSeed % 2) + 1;
  if(currentOpenLed != 1 && currentOpenLed != 2 &&  currentOpenLed != 3)
    currentOpenLed = 2;
  // open the required LED
  Clear_AllLeds();
  switch(currentOpenLed){
    case 1:
      Set_RedLed();
      break;
    case 2:
      Set_GreenLed();
      break;
  }
  return currentOpenLed;
}
```

GlobalConfiq:

This file has all the variables that are being used in all files such as the array of the game difficulty, attempt, and score variables.



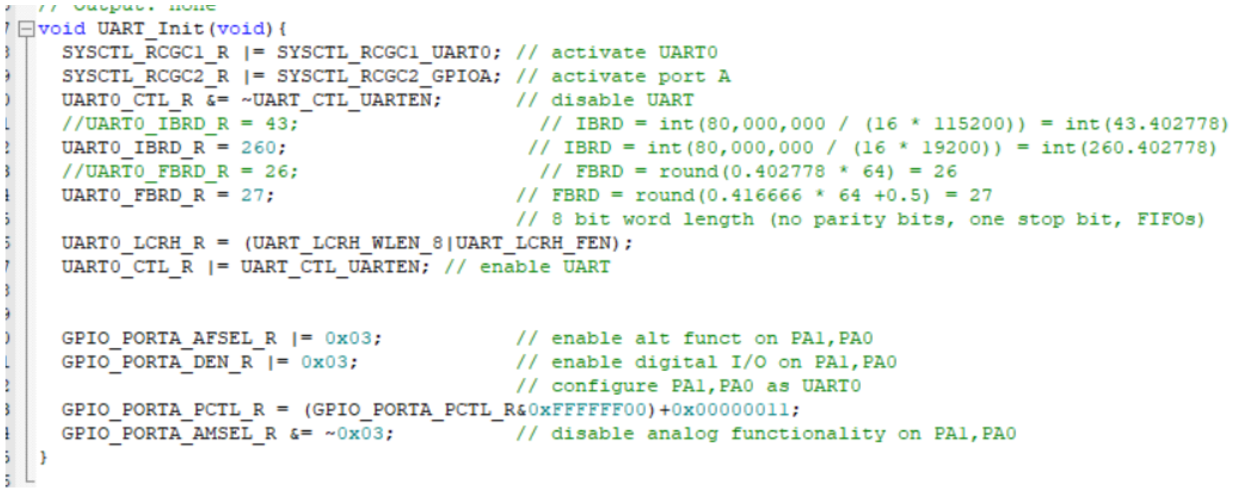
```
const unsigned long CyclesPerSec = 80000000;
const double GameDiffTimes[] = {1.5, 1.0, 0.5};
const double HoldDelay = 1.75;

short int GameDiff = 0;
short int score = 0;
short int attempt = 0;

short int LDR_Threshold = 1000;
volatile unsigned int isLight = 0;
```

UART:

This file effectively implements UART communication and provides utility functions for handling data in various formats. It is well-suited for debugging and data display in embedded systems.

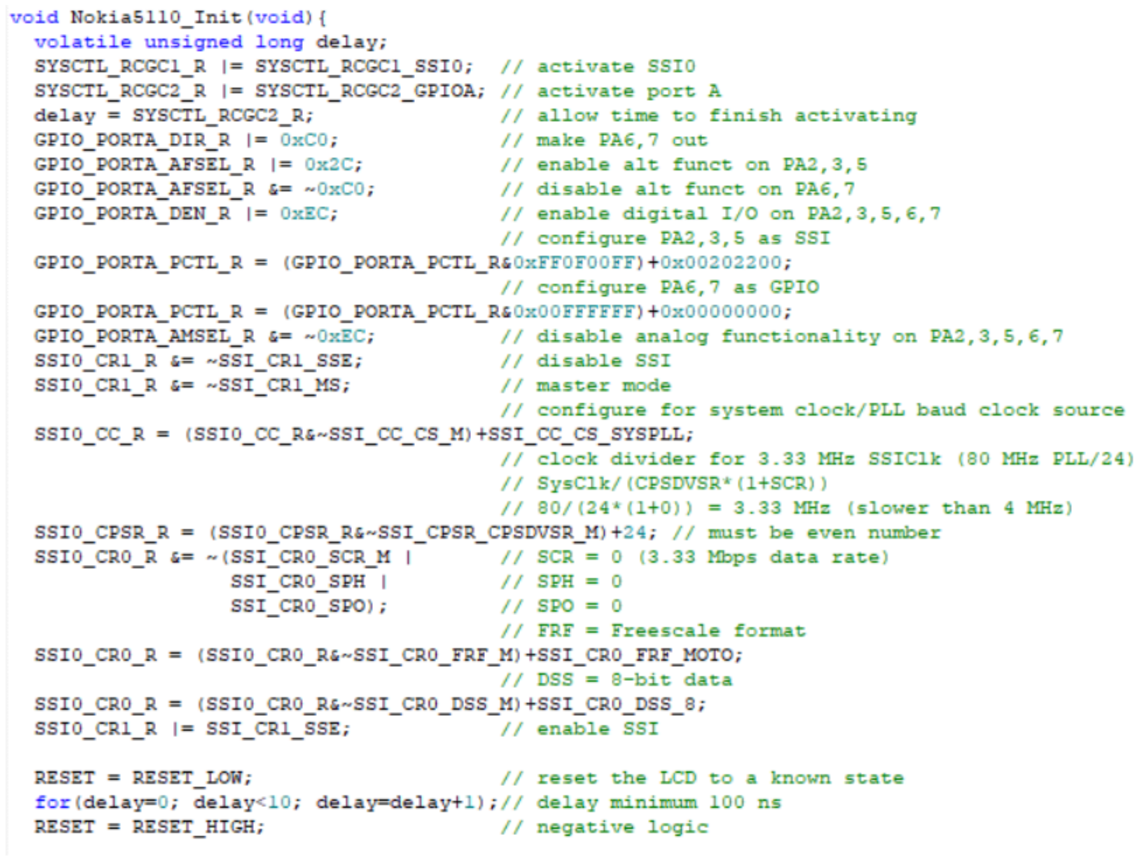
Enhancements in documentation, modularity, and testing can further improve its robustness and maintainability.

The `UART_Init()` function initializes the UART0 module to communicate at a baud rate of 19200. It activates UART0 and Port A, configures the baud rate parameters (IBRD and FBRD), and sets the data format to 8-bit word length, no parity, one stop bit, and enables FIFOs. It also configures pins PA0 (U0Rx) and PA1 (U0Tx) for UART functionality by enabling their alternate functions and disabling their analog mode. Finally, it enables the UART by setting the UARTEN bit in the control register.



```
// Output: none
void UART_Init(void){
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_UART0; // activate UART0
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate port A
  UART0_CTL_R &= ~UART_CTL_UARTEN;      // disable UART
  //UART0_IBRD_R = 43;                    // IBRD = int(80,000,000 / (16 * 115200)) = int(43.402778)
  UART0_IBRD_R = 260;                    // IBRD = int(80,000,000 / (16 * 19200)) = int(260.402778)
  //UART0_FBRD_R = 26;                    // FBRD = round(0.402778 * 64) = 26
  UART0_FBRD_R = 27;                    // FBRD = round(0.416666 * 64 +0.5) = 27
                                        // 8 bit word length (no parity bits, one stop bit, FIFOs)
  UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);
  UART0_CTL_R |= UART_CTL_UARTEN; // enable UART


  GPIO_PORTA_AFSEL_R |= 0x03;           // enable alt funct on PA1,PA0
  GPIO_PORTA_DEN_R |= 0x03;             // enable digital I/O on PA1,PA0
                                        // configure PA1,PA0 as UART0
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFFFFFF00)+0x00000011;
  GPIO_PORTA_AMSEL_R &= ~0x03;          // disable analog functionality on PA1,PA0
}
```

Nokia5110_init:



```
void Nokia5110_Init(void){
  volatile unsigned long delay;
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_SSI0;  // activate SSI0
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate port A
  delay = SYSCTL_RCGC2_R;               // allow time to finish activating
  GPIO_PORTA_DIR_R |= 0xC0;             // make PA6,7 out
  GPIO_PORTA_AFSEL_R |= 0x2C;           // enable alt funct on PA2,3,5
  GPIO_PORTA_AFSEL_R &= ~0xC0;          // disable alt funct on PA6,7
  GPIO_PORTA_DEN_R |= 0xEC;             // enable digital I/O on PA2,3,5,6,7
                                        // configure PA2,3,5 as SSI
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFF0F00FF)+0x00202200;
                                        // configure PA6,7 as GPIO
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0x00FFFFFF)+0x00000000;
  GPIO_PORTA_AMSEL_R &= ~0xEC;          // disable analog functionality on PA2,3,5,6,7
  SSI0_CR1_R &= ~SSI_CR1_SSE;           // disable SSI
  SSI0_CR1_R &= ~SSI_CR1_MS;            // master mode
                                        // configure for system clock/PLL baud clock source
  SSI0_CC_R = (SSI0_CC_R&~SSI_CC_CS_M)+SSI_CC_CS_SYSPLL;
                                        // clock divider for 3.33 MHz SSIClk (80 MHz PLL/24)
                                        // SysClk/(CPSDVSR*(1+SCR))
                                        // 80/(24*(1+0)) = 3.33 MHz (slower than 4 MHz)
  SSI0_CPSR_R = (SSI0_CPSR_R&~SSI_CPSR_CPSDVSR_M)+24; // must be even number
  SSI0_CR0_R &= ~(SSI_CR0_SCR_M |       // SCR = 0 (3.33 Mbps data rate)
                  SSI_CR0_SPH |         // SPH = 0
                  SSI_CR0_SPO);         // SPO = 0
                                        // FRF = Freescale format
  SSI0_CR0_R = (SSI0_CR0_R&~SSI_CR0_FRF_M)+SSI_CR0_FRF_MOTO;
                                        // DSS = 8-bit data
  SSI0_CR0_R = (SSI0_CR0_R&~SSI_CR0_DSS_M)+SSI_CR0_DSS_8;
  SSI0_CR1_R |= SSI_CR1_SSE;            // enable SSI

  RESET = RESET_LOW;                    // reset the LCD to a known state
  for(delay=0; delay<10; delay=delay+1);// delay minimum 100 ns
  RESET = RESET_HIGH;                   // negative logic
```

Steps in Initialization:

1. **Activate SSI0 and Port A:**
   - Enables the clock for SSI0 and GPIO Port A.
2. **Configure GPIO Pins:**
   - PA2, PA3, and PA5 are set to alternate functions for SSI communication.
   - PA6 and PA7 are configured as GPIO outputs for reset (RESET) and chip select (CS).
3. **Disable Analog Functions:**
   - Disables analog functionality for PA2, PA3, PA5, PA6, and PA7.
4. **Configure SSI Settings:**

    - Sets SSI in master mode, Freescale SPI format, and an 8-bit data size.
    - Configures SSI clock to operate at 3.33 MHz.
5. **Reset the LCD:**
    - Sends a reset pulse to ensure the LCD is in a known state.
6. **Set Extended Instruction Mode:**
    - Configures the LCD to use the extended instruction set.
    - Adjusts contrast, temperature coefficient, and bias settings.
7. **Switch to Basic Instruction Mode:**
    - Switches back to the basic instruction set.
    - Configures the display for normal mode.