Complex Engineering Problem Final Design and Assessment (EE-354)

Project Title:

Designing a Programmable Microcontroller-Based Signal Generator using ATmega328P and AD9833, among other peripherals

Project Team

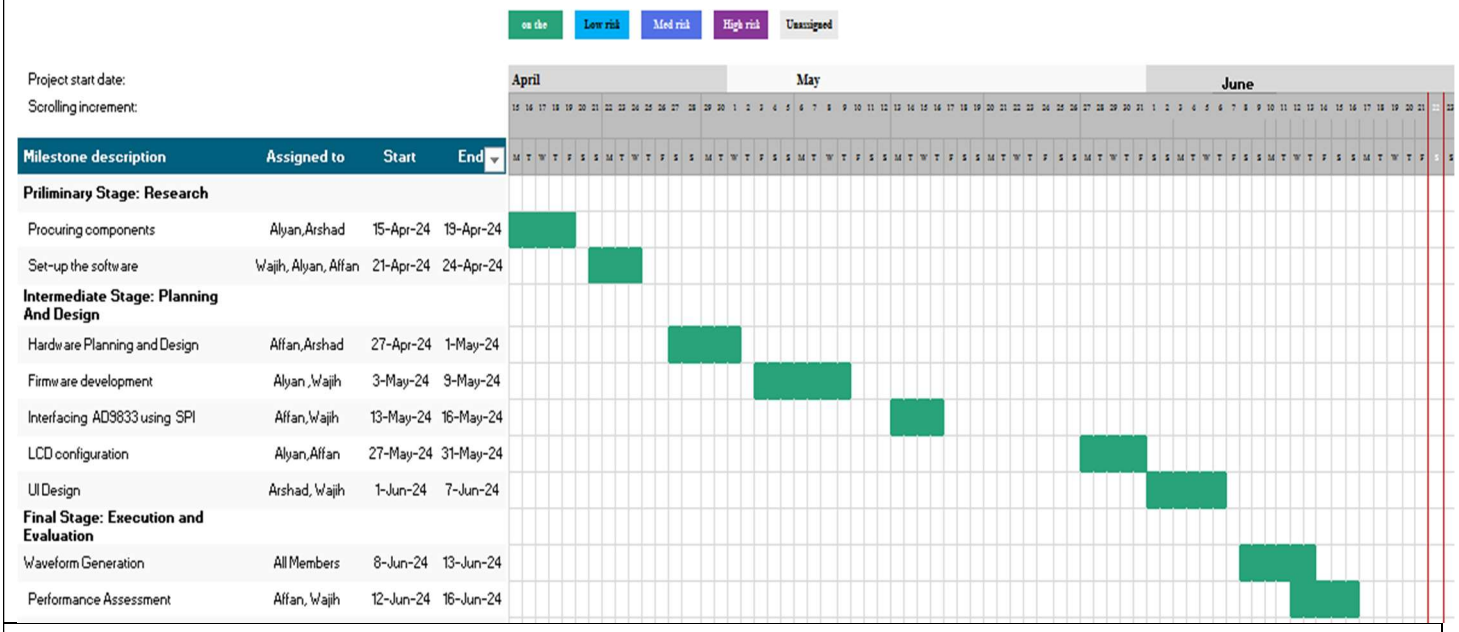
S.No.	Name	Roll No.	Section
1	Wajih Uddin	EE-21153	D
2	Syed Affan Hussain	EE-21156	D
3	Syed Alyan Ali	EE-21158	D
4	Muhammad Arshad Khan*	EE-21169	D

Summary of Progress:

The project aimed to design a programmable microcontroller-based signal generator using the ATmega328P and AD9833, along with other peripherals. Our team successfully interfaced the ATmega328P microcontroller with the AD9833 using the SPI protocol in Mode 2, ensuring data integrity through synchronized data transfer. The hardware connectivity was established with the AD9833 operating within its specified power range, and control registers were programmed to configure waveform generation parameters. We implemented a user-friendly interface, displaying generated waveform parameters on an LCD screen. The display effectively communicated the selected frequency and waveform type, allowing users to interpret and adjust settings as needed. Key technical achievements included configuring the SPI Control Register (SPCR) for SPI Mode 2 and programming the AD9833's control and frequency registers for precise waveform generation. The embedded C code was developed to manage SPI communication, AD9833 initialization, frequency setting, and waveform selection. PORTC was configured for user input to enable real-time adjustments to frequency and waveform type through push-button interactions. The integration of hardware and software components was executed, with seamless interaction between the microcontroller, AD9833, and input devices. The project concluded with successful testing and validation of the signal generator, confirming its ability to generate accurate waveforms across a range of frequencies.

Task Completed

PROJECT: Designing a Programmable Microcontroller-Based Signal Generator using ATmega328P and AD9833, among other pheripherals .



Interface with AD9833

Hardware Connectivity

The AD9833 operates with a power supply ranging from 2.3 V to 5.5 V. The Atmega328P microcontroller interfaces with the AD9833 using the SPI (Serial Peripheral Interface) protocol.

Communication Protocol:

SPI Protocol in Mode 2:

- **SPI Mode 2:** Data is written on the falling edge of the clock.
- **SPCR Configuration:** The SPI Control Register (SPCR) of Atmega328P is programmed to set SPI Mode 2 by configuring the clock polarity as required by the AD9833 datasheet.

Three-Wire SPI Interface:

- **SCLK (Serial Clock):** Generated by the microcontroller, provides the clock signal for synchronized data transfer.
- **SDATA (Serial Data):** Corresponds to SPI MOSI (Master Out Slave In), carries the 16-bit serial data word from the microcontroller to the AD9833.
- **FSYNC (Function Sync):** Corresponds to SPI SS (Slave Select), taken low to indicate that a new word is being loaded into the AD9833.

Control Registers:

The AD9833 features control registers that store configuration data for waveform generation:

- **Frequency Registers:** Store frequency data, 28 bits wide.
- **Phase Registers:** Store phase data.

The ATmega328P writes to these registers to configure the frequency, phase, and waveform type.

Parameter Display:

- **Communication:** The ATmega328P communicates the generated waveform parameters, such as the selected frequency (in Hz or kHz) and waveform type, to an LCD display.
- **User Interface:** The LCD screen presents this information in a clear and user-friendly manner, allowing the user to interpret the current waveform settings.

AD9833 Datasheet

The device is mainly controlled through use of a 16-bit control register. The control register configures the operation of the AD9833, depending on how each of its bits are utilized.

The Control Register:

D15, D14 (Register Selection)

- **00:** *Alters the content of the control register.*
- **01** or **10:** *Alters the content of the frequency registers.*
- **11:** *Alters the content of the phase registers.*

This means that the combination of bits D15 and D14 determines which type of register (control/frequency/phase) will be modified by subsequent operations.

D13 (B28 – 28-bit Frequency Mode)

- **B28 = 1:** Allows a complete word to be loaded into a frequency register in two consecutive writes. The first write contains the 14 least significant bits (LSBs) of the frequency word, and the next write contains the 14 most significant bits (MSBs).
- The first two bits of each 16-bit word define the frequency register to which the word is loaded. This allows for precise control over the frequency by splitting the 28-bit frequency word into two 14-bit words.

D8 (Reset)

It is a bit used to reset the AD9833 to a known state.

- **Reset = 1:** Resets internal registers to 0.
- **Reset = 0:** Disables reset.

D5, D3, D1 (Output Control)
As the name suggests, these bits control the type of the output waveform. Specific combinations of these bits define the output waveform type (sinusoidal, triangular, or square). This allows the user to select the desired waveform output.

D12, D11, D10, D9, D7, D4, D2, D0
These bits are not used for any functions specific to our application and should be set to 0 to avoid unintended behavior.

The Frequency and Phase Registers:
The AD9833 has two frequency registers (FREQ0 and FREQ1) and two phase registers (PHASE0 and PHASE1).

D11 (FSELECT – "Frequency Register Select")
Modifying the value of this bit allows the user to choose between the aforementioned two frequency registers for generating output.

- **FSELECT = 0:** Selects the FREQ0 register.
- **FSELECT = 1:** Selects the FREQ1 register.

D12 (PSELECT – "Phase Register Select")
Modifying the value of this bit similarly allows the user to choose between two different phase registers for generating output.

- **PSELECT = 0:** Selects the PHASE0 register.
- **PSELECT = 1:** Selects the PHASE1 register.

Using the Frequency Registers to Determine the Output Frequency:
The output frequency may be evaluated as;

$$FREQREG = f_{out} \times \frac{2^{28}}{f_{MCLK}}$$

Where:
$FREQREG$ is the value in the frequency register.
f_{out} is the desired output frequency.
f_{MCLK} is the master clock frequency.

This formula shows how the frequency register value translates into the actual output frequency based on the master clock, and was consequently used to evaluate our desired frequency values.

Embedded C code for AD9833

Initially, the SPI section of the microcontroller is configured by setting the control register to enable SPI as master with mode 2, where the clock polarity is set to 1.

To initialize the AD9833, the FSYNC bit of PORTB is cleared to activate the IC. Subsequently, the D13 (B28) and D8 (Reset) bits are set in the AD9833 Control Register. The MSB byte is transmitted first, followed by waiting for the transmission to finish. Then, the LBS byte is sent, and the transmission is awaited again. Finally, the FSYNC bit of PORTB is set to disable the IC.

Following this initial transmission, the next lines of code are executed to program the frequency register with a default frequency with a default frequency of 1000. The frequency is calculated by converting it to its decimal equivalent, using a multiplication factor provided in a preceding section of the report. The 14 LSBs of the 28-bit frequency register are transmitted (for FREQ0, OR-ed with 01), followed by the 14 MSBs. The transmission is concluded by returning to the main part of the code.

Next, code lines are executed to transmit the 12 bits of the phase register (for PHASE0, OR-ed with 1100). Following this, the D8 (Reset) bit in the AD9833 Control Register is cleared to disable the reset. Then, code lines are executed to set the waveform mode to sine. This involves configuring the control register bits D5 (OPBITEN), D3 (DIV2), and D1

(Mode) to select the desired waveform type. A switch-case statement is utilized to appropriately set these bits for sinusoidal, triangular, or square waveforms. Enumeration is employed to manage these waveform types, introducing a new variable type for flexibility. Several other actions are also performed. The program configures PORT C pins 3, 4, and 5 as inputs by clearing the corresponding bits in the data direction register (DDRC) with *DDRC &= ~(0x38)*. Initialization functions *AD_init()* and *lcd_init()* are called to set up the AD9833 and LCD, respectively. Key variables *x, freq_val*, and *str_temp* manage waveform and frequency settings. *freq_val = AD_getFrequency* retrieves the current frequency value. An *if condition* checks if *freq_val* is between 100 Hz and 100 kHz. Within this condition, further checks adjust the frequency up or down by 100 Hz using *AD_setFrequency()* when push buttons on PC5 or PC4 are pressed. The LCD is updated with the current frequency using *lcd_command()* and *lcd_print()* functions. Another check monitors PC3 for a button press, incrementing *x* to cycle through waveform modes (triangle, sine, or square), displaying the selected mode on the LCD. Once *x* reaches 3, it resets to 0, completing the cycle. The program remains in an infinite loop, continually updating the LCD with frequency and waveform information based on user interactions.

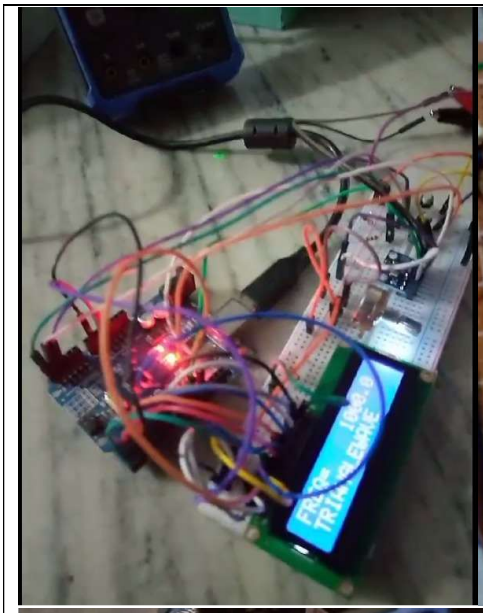
Complete Hardware-Software Integration

The hardware integration involved establishing a connection between the Atmega328P microcontroller and the AD9833 waveform generator through the SPI protocol in Mode 2, which provided synchronized data transmission. User interaction was facilitated through push-button inputs linked to PORTC, enabling real-time adjustments to waveform parameters such as frequency and waveform type. This user interface was integrated into the system's operational logic for responsive control. The software component of the project involved the development of an embedded C codebase to orchestrate interactions between the microcontroller and the waveform generator. Key functions such as *AD_write16*, *AD_setFrequency*, *AD_setPhase*, and *AD_setMode* are pivotal in configuring the AD9833 chip to generate specific waveform characteristics under user control.

The *AD_init* function initializes the necessary hardware configurations and sets default parameters for the signal generator upon startup. This includes configuring the SPI communication on the microcontroller and initializing the AD9833 with initial settings such as frequency of 1000 Hz, phase of 0 degrees, and sine wave mode. These settings are important as they dictate the initial operational state of the signal generator.

The *main* function continuously monitors the push-button inputs. If buttons connected to PC5 or PC4 are pressed, the *Ad_setFrequency* function adjusts the frequency up or down by 100 Hz increments, providing immediate feedback on an LCD screen through the *lcd_print* function. This interactive feature allows us to fine-tune the signal frequency. Similarly, the *AD_setMode* function is used to switch between different waveform types (sine, square, triangle) based on user input from PC3. This functionality is integrated with the LCD display to reflect the current elected waveform type. The LCD interface is initialized and managed through functions *lcd_init*, *lcd_command*, and *lcd_data*, which set up and control a standard 16x2 character LCD screen connected to PORTD and PORTB. This interface provides visual feedback on the current frequency setting and dynamically updates to reflect changes in waveform type selected by the user.

Results





Risk Assessment and Mitigation	
Risk Mitigation	

1. Design Complexity Risk:
   Risk Level: Medium
   Mitigation Strategy: To mitigate complexity-related issues early on, the project will adopt a modular approach, breaking down tasks into manageable modules.

2. Software Development Risk:
   Risk Level: Medium
   Adequate time allocation for software development and testing will be ensured Regular code reviews and testing protocols will be implemented to detect and address bug promptly.

3. Performance and Calibration Risk:
   Risk Level: High
   Calibration should be considered to uphold the signal generator's accuracy and precision over a range of frequencies and waveforms. Testing during various development stages will be conducted.

4. Power Outage during Experimentation Risk:
   Risk Level: Low

	To mitigate the risk of a potential power outage during experimentation, backup power sources will be readily available in the event of adaptor failure or power interruption.
References	

Analog Devices. (2019). Low Power, 12.65 mW, 2.3 V to 5.5 V, Programmable Waveform Generator. AD 9833 datasheet.

Analog Devices. (2003). AN-1070 Application Note. AD9833 programming.

Atmel. (2015). 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash. ATmega328P datasheet.

Ioan, M. (2010). Design of a Function Generator using Direct Digital Synthesis (DDS) Technology and PIC16F877A Microcontroller. Annals of the Oradea University, 19(2).

Rego, P. V. (2012). Integrating 8-bit Micro-controllers in Ada. Ada User Journal, 3(4).

Rospawan, A., Simatupang, J.W., & Purnama, I. (2019). A simple, cheap and precise microcontroller-based DDS based function generator. Journal of Electrical and Electronics Engineering, 3(2), 118-121.

Yashas, S.R., Kulkarni, S., & Kumara, B. (2019). Waveform Generation using Direct Digital Synthesis (DDS) Technique. International Research Journal of Engineering and Technology (IRJET), 6(11).

Ionescu, C. (202). AVR-1000b: Getting Started with Writing C-Code for AVR MCUs. Microchip Technology, Datasheet module.

Ada. W.F. (2012). Low distortion signal generator based on direct digital synthesis of ADC characteriszation. Acta IMEKO, 1(1), 59-64.

Attachments:	Attach print of your final C code along with any header files

```
#include <avr/io.h>
#include <stdint.h>
#include <stdlib.h>
#include <util/delay.h>
#define DATA 3
#define CLK 5
#define FSYNC 2

float _freq;

typedef enum mode
{
    MODE_SINE,
    MODE_SQUARE,
    MODE_TRIANGLE,
} mode_t;

void AD_write16(uint16_t data)
{
    PORTB &= ~(1<<FSYNC);
    SPDR = (data>>8);
    while (!(SPSR & (1<<SPIF)))
        ;
    SPDR = (data&0xFF);
    while (!(SPSR & (1<<SPIF)))
        ;
    PORTB |= (1<<FSYNC);
}

void AD_setFrequency(float freq)
{
      _freq= freq;
    uint32_t _regFreq = freq*10.73742 + 0.5;
    AD_write16(0x4000 | (uint16_t)(_regFreq & 0x3FFF));
    AD_write16(0x4000 | (uint16_t)((_regFreq>>14) & 0x3FFF));
}

const float AD_getFrequency()
{
    return _freq;
}

void AD_setPhase(uint16_t phase)
{
    uint32_t _regPhase = (51.2*phase)/45 + 0.5;
    AD_write16(0xC000 | (uint16_t)(_regPhase & 0xFFF));
}

void AD_setMode(mode_t mode)
{
    switch(mode)
    {
        case(MODE_SINE):
            AD_write16(0x2000);
            break;
        case(MODE_SQUARE):
            AD_write16(0x2028);
            break;
        case(MODE_TRIANGLE):
            AD_write16(0x2002);
            break;
    }
}
```

```
void AD_init(void)
{
    DDRB = (1<<DATA) | (1<<CLK) | (1<<FSYNC);
    PORTB &= ~(1<<CLK);
    PORTB |= (1<<FSYNC);
    SPCR = 0x59;
    AD_write16(0x2100);
    AD_setFrequency(1000);
    AD_setPhase(0);
    AD_write16(0x2000);
    AD_setMode(MODE_SINE);
}

void lcd_command(unsigned char cmnd)
{
    PORTD = cmnd;
    PORTB &= ~(1<<PB0);
    PORTB |= (1<<PB1);
    _delay_us(1);
    PORTB &= ~(1<<PB1);
    _delay_us(100);
}

void lcd_data(unsigned char data)
{
    PORTD = data;
    PORTB |= (1<<PB0);
    PORTB |= (1<<PB1);
    _delay_us(1);
    PORTB &= ~(1<<PB1);
    _delay_us(100);
}

void lcd_init()
{
    DDRD = 0xFF;
    DDRB |= 0x03;
    PORTB &= ~(1<<PB1);
    _delay_us(2000);
    lcd_command(0x38);
    lcd_command(0x0C);
    lcd_command(0x01);
    _delay_us(2000);
}

void lcd_print(char* StringPtr)
{
    unsigned char i =0;
    while (StringPtr[i]!=0)
    {
        lcd_data(StringPtr[i]);
        i++;
    }
}

int main(void)
{
    DDRC &= ~(0x38);
    AD_init();
    lcd_init();
    _delay_ms(1000);
    int x = 0;
```

```
    float freq_val = 0;
    char str_temp[10];
    while(1)
    {

        freq_val = AD_getFrequency();
        if((freq_val> 100)&& (freq_val< 100000))
        {
            if (PINC & (1<<PC5))
            {
                AD_setFrequency(freq_val + 100);
            }
            if (PINC & (1<<PC4))
            {
                AD_setFrequency(freq_val - 100);
            }
        }

        lcd_command(0x02);
        _delay_ms(500);
        lcd_print("FREQ=");
        dtostrf(freq_val,10,1,str_temp);
        lcd_print(str_temp);

        if(PINC & (1<<PC3))
        {
            x += 1;
            switch(x)
            {
            case(1):
                AD_setMode(MODE_TRIANGLE);
                lcd_command(0xC0);
                lcd_print("TRIANGLEWAVE");
                break;
            case(2):
                AD_setMode(MODE_SINE);
                lcd_command(0xC0);
                lcd_print("SINE    ");
                break;
            case(3):
                AD_setMode(MODE_SQUARE);
                lcd_command(0xC0);
                lcd_print("SQUARE  ");
                x = 0;
                break;
            }
        }

    }

    return 0;
}
```

Skill(s) to be assessed	Extent of Achievement			
	50%	65%	80%	100%
Application of theoretical knowledge for analysis	Unable to relate and recall any theory	Partially relates and recalls the theory	Recalls theoretical knowledge and relates it to the task without major mistakes	Recalls theoretical knowledge and relates it to the task with no mistake
Updates in Resource identification and allocation, task identification and scheduling, risk assessment and management	No updates or revision	----------	----------	Reasonable updates and revision presented
Realisation of design, testing and validation	Unable to synthesize and also unable to test/validate	Able to synthesize partially and able to test/validate OR Able to synthesize but partially able to test/validate	Able to synthesize and test/validate without major mistakes	Able to synthesize and test/validate without any mistake
Reporting hardware testing results in verbal and graphical manner	No results reported	Results reported but missing major deliverables	Results reported with minor issues	Results reported, meeting all deliverables

Skill(s) to be assessed	Extent of Achievement			
	50%	65%	80%	100%
Application of theoretical knowledge for analysis	Unable to relate and recall any theory	Partially relates and recalls the theory	Recalls theoretical knowledge and relates it to the task without major mistakes	Recalls theoretical knowledge and relates it to the task with no mistake
Updates in Resource identification and allocation, task identification and scheduling, risk assessment and management	No updates or revision	----------	----------	Reasonable updates and revision presented
Realisation of design, testing and validation	Unable to synthesize and also unable to test/validate	Able to synthesize partially and able to test/validate OR Able to synthesize but partially able to test/validate	Able to synthesize and test/validate without major mistakes	Able to synthesize and test/validate without any mistake
Reporting hardware testing results in verbal and graphical manner	No results reported	Results reported but missing major deliverables	Results reported with minor issues	Results reported, meeting all deliverables

Skill(s) to be assessed	Extent of Achievement			
	50%	65%	80%	100%
Application of theoretical knowledge for analysis	Unable to relate and recall any theory	Partially relates and recalls the theory	Recalls theoretical knowledge and relates it to the task without major mistakes	Recalls theoretical knowledge and relates it to the task with no mistake
Updates in Resource identification and allocation, task identification and scheduling, risk assessment and management	No updates or revision	----------	----------	Reasonable updates and revision presented
Realisation of design, testing and validation	Unable to synthesize and also unable to test/validate	Able to synthesize partially and able to test/validate OR Able to synthesize but partially able to test/validate	Able to synthesize and test/validate without major mistakes	Able to synthesize and test/validate without any mistake
Reporting hardware testing results in verbal and graphical manner	No results reported	Results reported but missing major deliverables	Results reported with minor issues	Results reported, meeting all deliverables

Skill(s) to be assessed	Extent of Achievement			
	50%	65%	80%	100%
Application of theoretical knowledge for analysis	Unable to relate and recall any theory	Partially relates and recalls the theory	Recalls theoretical knowledge and relates it to the task without major mistakes	Recalls theoretical knowledge and relates it to the task with no mistake
Updates in Resource identification and allocation, task identification and scheduling, risk assessment and management	No updates or revision	----------	----------	Reasonable updates and revision presented
Realisation of design, testing and validation	Unable to synthesize and also unable to test/validate	Able to synthesize partially and able to test/validate OR Able to synthesize but partially able to test/validate	Able to synthesize and test/validate without major mistakes	Able to synthesize and test/validate without any mistake
Reporting hardware testing results in verbal and graphical manner	No results reported	Results reported but missing major deliverables	Results reported with minor issues	Results reported, meeting all deliverables