555 Timer Design Assistant

Programming for Electronics – 663-104

07/30/25

Submitted to: David Duckert, Instructor

Submitted by: Rudolph Hofmeister

Problem Layout

The 555 timer is a very useful tool in the electronics world. Its main purpose is to create a square wave, either as a pulse (monostable), or a continuous wave (astable). For each case, the user will need to choose certain resistor and capacitor values based on the pulse width, frequency, or duty cycle you need.

Although the math to choose these values is fairly simple, for each circuit there are unlimited possibilities of resistors and capacitors values. Doing these calculations multiple times to get "clean" combinations can be time-consuming, error-prone, and overall annoying.

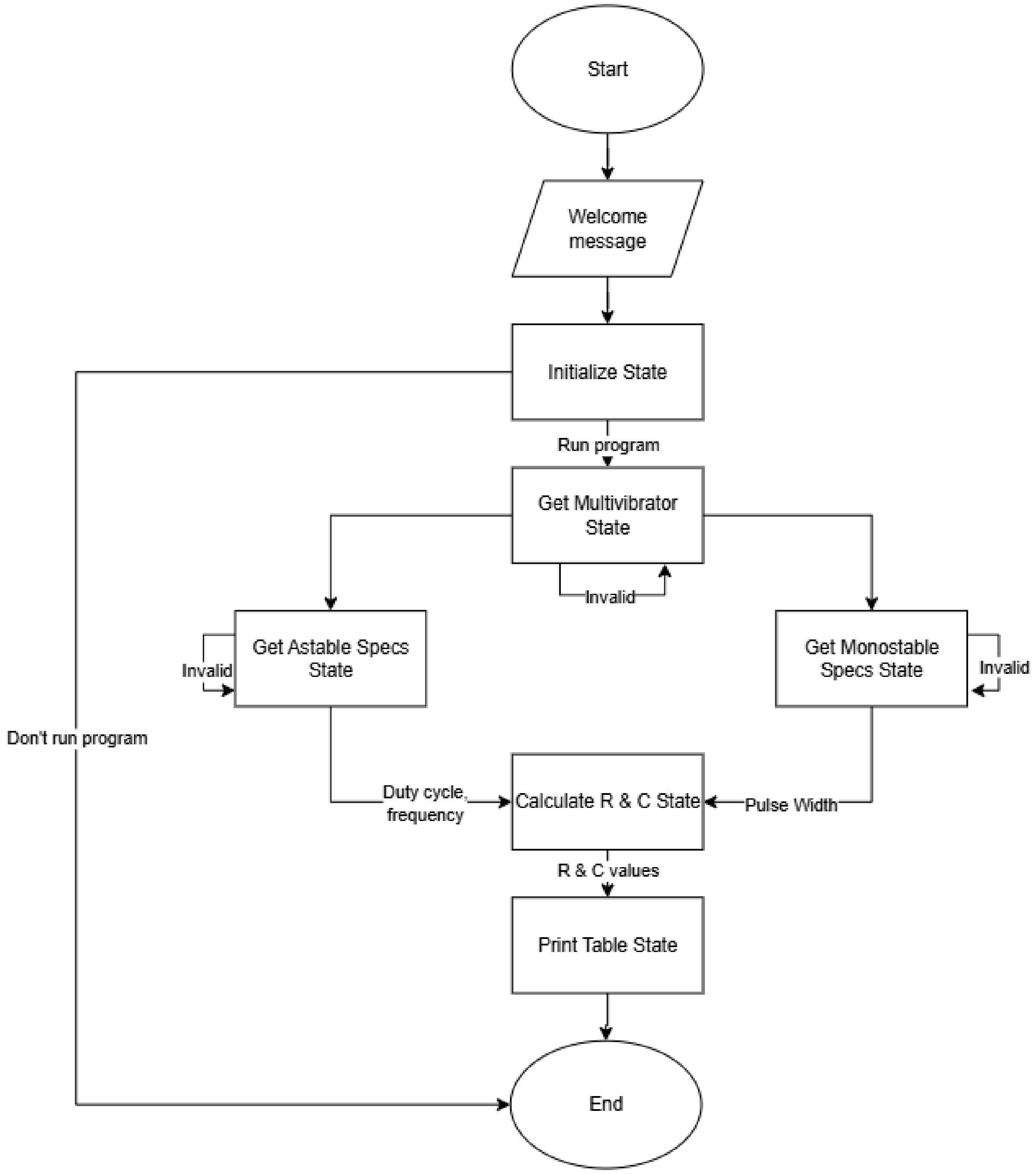
This is where a calculator would be very handy. All the user has to do is enter their intention for the circuit (astable or monostable), and the criteria for each (pulse width, frequency, or duty cycle). The program then computes a nice list of all the combinations, and prints them out for the user to choose. This would both save time, and ensure accurate results for all circuits.

Approach to Problem

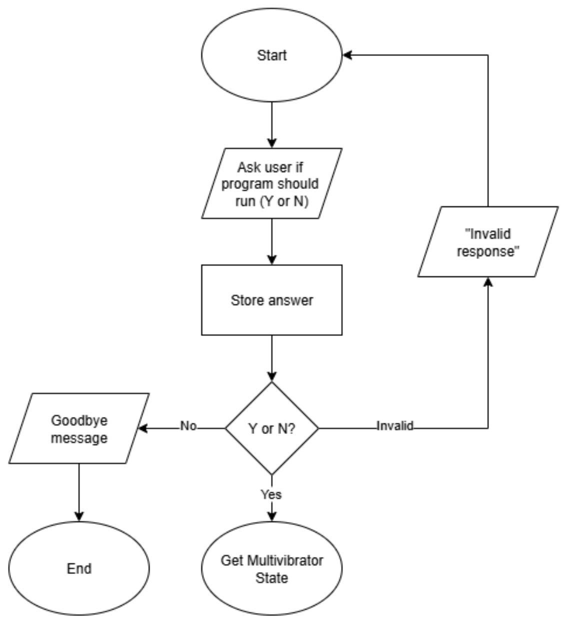
The program would work as follows:

1. **Welcome Message:** Explain the intention of the program
2. **Prompt to Run:** The user is asked if they want to run the program
3. **Mode Selection:** The user chooses if they are setting up an astable, or monostable circuit
4. **User Specification:** Depending on the mode, the user is asked for
    - Frequency and duty cycle (astable)
    - Pulse width (monostable)
5. **Validation**: Each input is checked. If the input is invalid, it asks the user for clarification
6. **Calculation**: The program computes resistor values based on a list of typical capacitor values. If the resistor value is a reasonable value, it is added to the list.
7. **Output**: A clear table or possible resistor and capacitor values are printed for the user

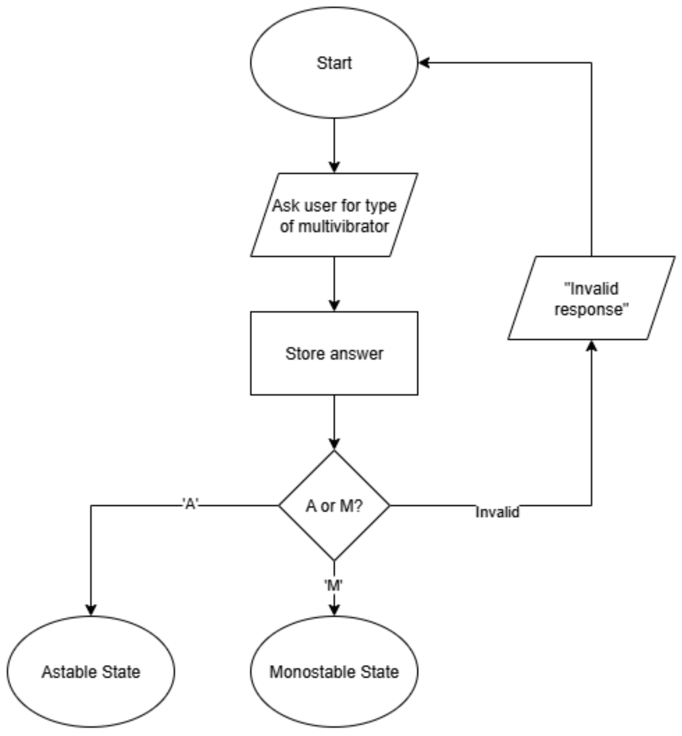
State Machine Diagram



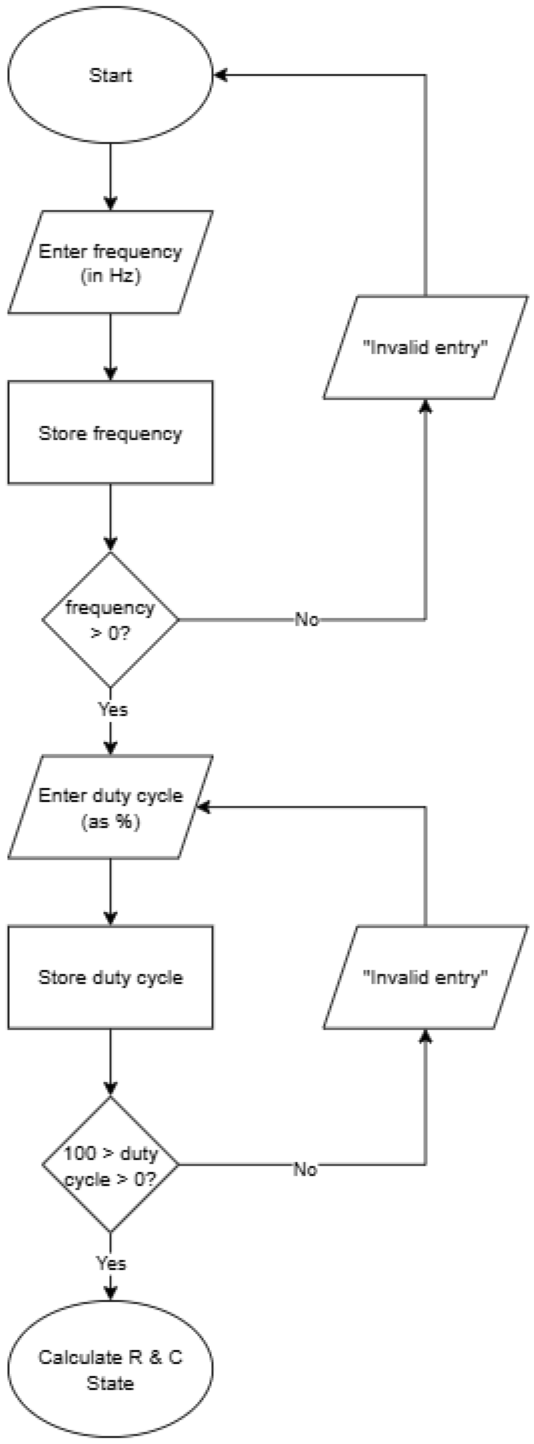
Prompt to Run State



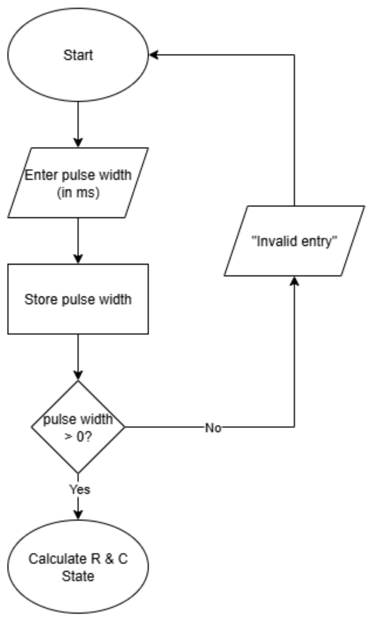
Mode Selection State



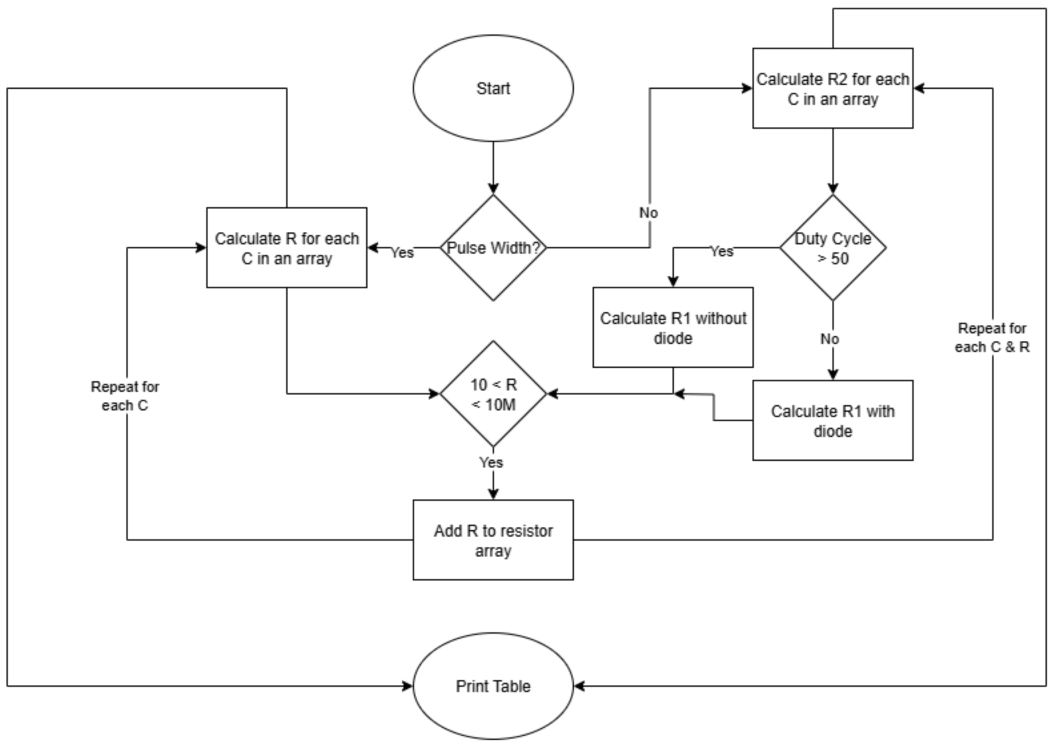
Get Astable Specs State



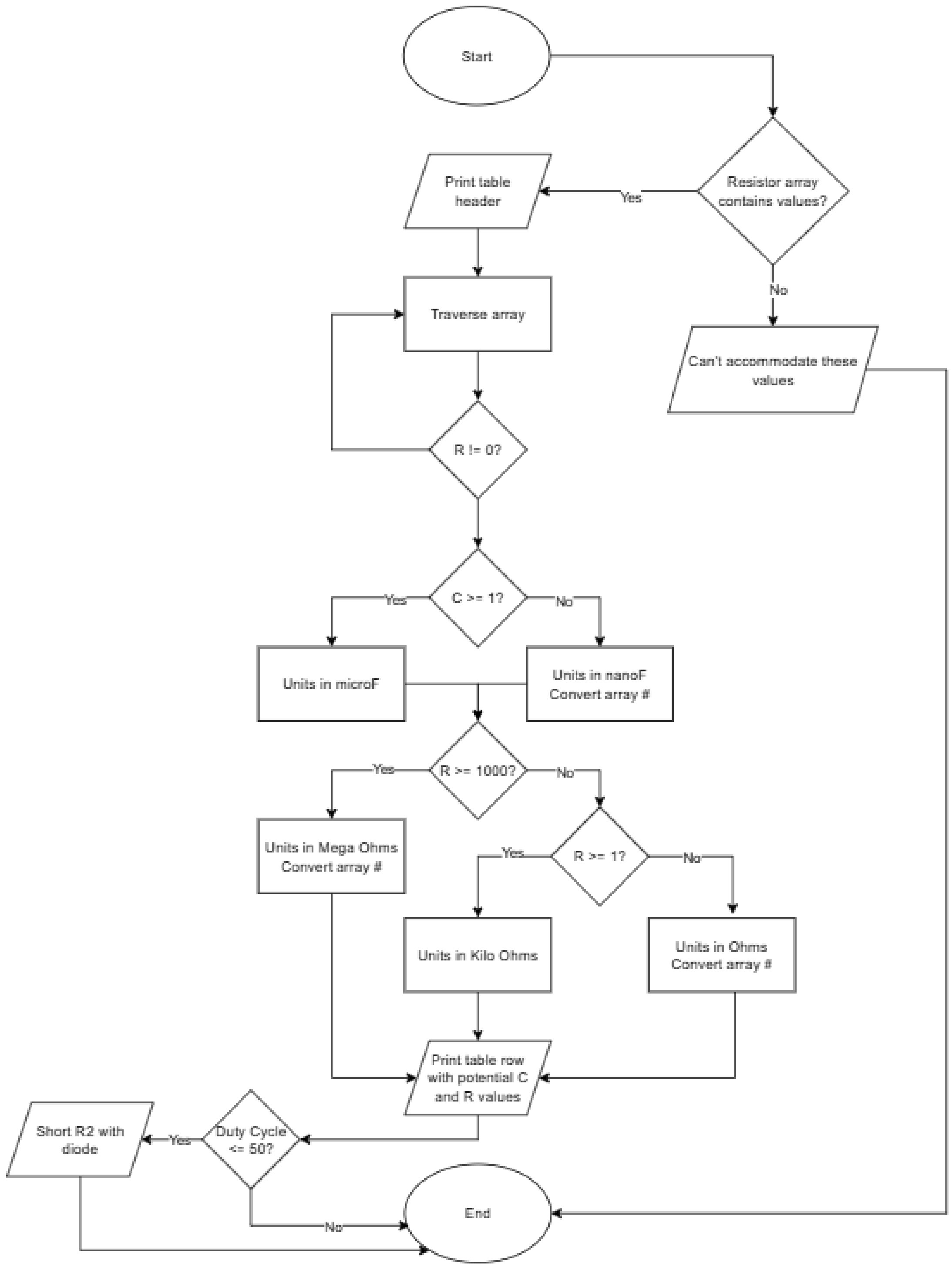
Get Monostable Specs State



Calculate State



Print Table State


Start
Resistor array contains values?
Yes
No
Print table header
Can't accommodate these values
Traverse array
R != 0?
C >= 1?
Yes
No
Units in microF
Units in nanoF Convert array #
R >= 1000?
Yes
No
Units in Mega Ohms Convert array #
R >= 1?
Yes
No
Units in Kilo Ohms
Units in Ohms Convert array #
Print table row with potential C and R values
Duty Cycle <= 50?
Yes
No
Short R2 with diode
End

Code

```
/*
Calculate the needed capacitor and resistor values for monostable or astable 555 timer applications
Rudolph Hofmeister 07/02/2025
*/

#define _CRT_SECURE_NO_WARNINGS // Tells cisual studios to ignore unsafe scanf functions
#include <stdio.h>

#define ASTABLE_CONSTANT 0.69
#define MONOSTABLE_CONSTANT 1.10

/* Program Outputs */
double typical_c_uF[] = { 0.001, 0.0055, 0.01, 0.1, 0.22, 0.47, 1, 10, 22, 47, 100 };
double r_one_ko[11];
double r_two_ko[11];

/* States for state machine (from textbook) */
typedef enum
{
	initial_state,
	get_multivibrator_state,
	astable_state,
	monostable_state,
	calculate_state,
	print_table_state,
	done_state
}
states_t;

states_t current_state = initial_state;

/* State function prototypes */
void initial(void);
void get_multivibrator(void);
double astable_frequency(void);
double astable_duty_cycle(void);
double monostable_pulse_width(void);
void calculate_astable(double f, double dc);
void calculate_monostable(double pw);
void print_table_astable(double dc);
void print_table_monostable(void);

/* Program function prototyopes */
char get_unit_c(double *p_c_uF);
char get_unit_r(double *p_r_ko);
int contains_values(double r_one[]);

int main(void)
{
```

```
/* Program Inputs */
double frequency = -1;
double duty_cycle = -1;
double pulse_width = -1;

/* Welcome message */
printf("Welcome to the 555 Timer Design Assistant!\n");

while (current_state != done_state)
{
        switch (current_state)
        {
                case initial_state:
                        /* Find out if program should run */
                        initial();
                        break;

                case get_multivibrator_state:
                        /* Get the type of multivibrator from the user */
                        get_multivibrator();
                        break;

                case astable_state:
                        /* Get astable specs from user */
                        while (frequency == -1)
                        {
                                frequency = astable_frequency();
                        }
                        while (duty_cycle == -1)
                        {
                                duty_cycle = astable_duty_cycle();
                        }
                        current_state = calculate_state;
                        break;

                case monostable_state:
                        /* Get monostable specs from user */
                        while (pulse_width == -1)
                        {
                                pulse_width = monostable_pulse_width();
                        }
                        current_state = calculate_state;
                        break;

                case calculate_state:
                        if (pulse_width == -1)
                        {
                                /* Put together arrays of possible R1 and R2 values */
                                calculate_astable(frequency, duty_cycle);
```

```
				}
				else
				{
					/* Put together an array of possible R values */
					calculate_monostable(pulse_width);
				}
				current_state = print_table_state;
				break;

			case print_table_state:
				if (pulse_width == -1)
				{
					/* Print table of possible combinations of R1, R2, & C */
					print_table_astable(duty_cycle);
				}
				else
				{
					/* Print table of possible combinations of R & C */
					print_table_monostable();
				}
				current_state = done_state;
				break;
		}
	}

	/* Program end message */
	printf("\n\nThank you!\n\n");

	return(0);
}

void initial(void)
{
	char run;

	/* Ask user if they would like to run the program */
	printf("\nWould you like to run the program? (Y or N): ");

	/* Store response */
	scanf(" %c", &run); // Space before %c tells scanf to skip any leftover whitespace characters (when the
user presses enter)

	/* Determine if they would like to start/end program, or if response was unclear */
	switch(run)
	{
		case 'Y':
		case 'y':
			current_state = get_multivibrator_state;
			break;
			case 'N':
			case 'n':
				current_state = done_state;
				break;

			default:
				printf("\nInvalid entry, please try again\n");
				while (getchar() != '\n'); // Clears invalid input so the loop won't repeat multiple times
(from chat gpt)
				break;
		}
}

void get_multivibrator(void)
{
		char type_of_multivibrator;

		/* Ask user to enter the type of multivibrator they are using */
		printf("\nAre you using the 555 timer as an astable, or monostable multivibrator?\n");
		printf("Type A for astable or M for monostable: ");

		/* Store response */
		scanf(" %c", &type_of_multivibrator); // Space before %c tells scanf to skip any leftover whitespace
characters, this is due to back-to-back input in the scanf function (from chat gpt)

		/* Decide which type of multivibrator they chose */
		switch (type_of_multivibrator)
		{
			case 'A':
			case 'a':
				current_state = astable_state;
				break;

			case 'M':
			case 'm':
				current_state = monostable_state;
				break;

			default:
				printf("\nInvalid entry, please try again\n");
				while (getchar() != '\n'); // Clears invalid input so the loop won't repeat multiple times
(from chat gpt)
				break;
		}
}

double astable_frequency(void)
{
	double f;

	/* Get frequency from user */
	printf("\nPlease enter frequency (in Hz): ");

	/* Store frequency */
	scanf("%lf", &f);

	/* Check a vaild frequency was entered */
	if (f > 0)
	{
		return(f);
	}
	else
	{
		printf("\nInvalid frequency entered, please try again");
		while (getchar() != '\n'); // Clears invalid input so the loop won't repeat multiple times (from
chat gpt)
		return(-1);
	}
}

double astable_duty_cycle(void)
{
	double dc;

	/* Get duty cycle from user */
	printf("\nPlease enter duty cycle (as a %%): ");

	/* Store duty cycle */
	scanf("%lf", &dc);

	/* Check a vaild duty cycle was entered */
	if ((dc > 0) && (dc < 100))
	{
		return(dc);
	}
	else
	{
		printf("\nInvalid duty cycle entered, please try again");
		while (getchar() != '\n'); // Clears invalid input so the loop won't repeat multiple times (from
chat gpt)
		return(-1);
	}
}

double monostable_pulse_width(void)
{
	double pw;
	/* Get pulse width from user */
	printf("\nPlese enter pulse width (in ms): ");

	/* Store pulse width */
	scanf("%lf", &pw);

	/* Check a vaild pulse width was entered */
	if (pw > 0)
	{
		return(pw);
	}
	else
	{
		printf("\nInvalid pulse width entered, please try again");
		while (getchar() != '\n'); // Clears invalid input so the loop won't repeat multiple times
		return(-1);
	}
}

void calculate_astable(double f, double dc)
{
	double r_one;
	double r_two;

	/* Formula: time low = (% of time low) * period */
	double time_low = (1 - (dc / 100)) * (1 / f);

	/* Calculate resistor values for each capacitor in the array */
	for (int i = 0; i < 11; i++)
	{
		/* Formula: Low time = 0.69R2C */
		r_two = time_low / (ASTABLE_CONSTANT * typical_c_uF[i]);

		if (dc > 50)
		{
			/* Formula: Period = 0.69(R1 + 2R2)C */
			r_one = ((1 / f) / (ASTABLE_CONSTANT * typical_c_uF[i])) - (2 * r_two);
		}
		else if (dc <= 50)
		{
			/* Formula: Period = 0.69(R1 + R2)C */
			r_one = ((1 / f) / (ASTABLE_CONSTANT * typical_c_uF[i])) - r_two;
		}

		/* Convert to Kilo Ohms */
		r_one *= 1000;
		r_two *= 1000;
```

```
                /* Check if that resistor value is feasible (recomended between 1K and 10M) */
                if ((r_one >= 1) && (r_two >= 1) && (r_one <= 10000) && (r_two <= 10000))
                {
                        r_one_ko[i] = r_one;
                        r_two_ko[i] = r_two;
                }
                else
                {
                        r_one_ko[i] = 0;
                        r_two_ko[i] = 0;
                }
        }
}

void calculate_monostable(double pw)
{
        double r;

        /* Calculate resistor value for each capacitor in the array */
        for (int i = 0; i < 11; i++)
        {
                /* Formula: pulse width = 1.10R1C */
                r = pw / (MONOSTABLE_CONSTANT * typical_c_uF[i]);

                /* Check if that resistor value is feasible (recomended between 1K and 10M) */
                if ((r >= 1) && (r <= 10000))
                {
                        r_one_ko[i] = r;
                }
                else
                {
                        r_one_ko[i] = 0;
                }
        }
}

void print_table_astable(double dc)
{
        /* Check to see if there are values to print in the table */
        if (contains_values(r_one_ko) == 1)
        {
                char unit_c;
                char unit_r_one;
                char unit_r_two;

                /* Print table header */
                printf("\n          C                    |             R1                  |             R2                    ");
                printf("\n-------------------------------------------------------------------------------");
```

```
		for (int i = 0; i < 11; i++)
		{
			/* Only print table row if values are stored in R1 and R2 */
			if ((r_one_ko[i] != 0) && (r_two_ko[i] != 0))
			{
				/* Decide if capacitor value would best be measured in micro or nano farads
*/
				unit_c = get_unit_c(&typical_c_uF[i]);

				/* Decide if R1 & R2 value would best be measured in Mega, Kilo, or ohms */
				unit_r_one = get_unit_r(&r_one_ko[i]);
				unit_r_two = get_unit_r(&r_two_ko[i]);

				printf("\n	%.1f%cf		|		%.1f%cohm	|
%.1f%cohm	", typical_c_uF[i], unit_c, r_one_ko[i], unit_r_one, r_two_ko[i], unit_r_two);

printf("\n--------------------------------------------------------------------------------");
			}
		}

		if (dc <= 50)
		{
			printf("\n\nNOTE: You will need to short R2 with a diode");
		}
	}
	else
	{
		printf("\nSorry, we can't accommodate these values.");
	}
}

void print_table_monostable(void)
{
	if (contains_values(r_one_ko) == 1)
	{
		char unit_c;
		char unit_r;

		/* Print table header */
		printf("\n	C		|	R	");
		printf("\n------------------------------------------------");

		for (int i = 0; i < 11; i++)
		{
			/* Only print table row if a value is stored in R1 */
			if (r_one_ko[i] != 0)
			{
				/* Decide if capacitor value would best be measured in micro or nano farads
*/
```

```
                    unit_c = get_unit_c(&typical_c_uF[i]);

                    /* Decide if resistor value would best be measured in Mega, Kilo, or ohms */
                    unit_r = get_unit_r(&r_one_ko[i]);

                    printf("\n        %.1f%cf          |          %.1f%cohm        ",
typical_c_uF[i], unit_c, r_one_ko[i], unit_r);
                    printf("\n-----------------------------------------------");
                }
            }
        }
        else
        {
            printf("\nSorry, we can't accommodate these values.");
        }
}

char get_unit_c(double *p_c_uF)
{
        /* Decide if capacitor value would best be measured in micro or nano farads */
        if (*p_c_uF >= 1)
        {
            return ('u');
        }
        else if (*p_c_uF >= 0.001)
        {
            *p_c_uF *= 1000;
            return('n');
        }
}

char get_unit_r(double *p_r_ko)
{
        /* Decide if the R value would best be measured in Mega, Kilo, or ohms */
        if (*p_r_ko >= 1000)
        {
            *p_r_ko /= 1000;
            return('M');
        }
        else if (*p_r_ko >= 1)
        {
            return('K');
        }
        else if (*p_r_ko >= 0.001)
        {
            *p_r_ko *= 1000;
            return('\0'); // No unit, just regular ohms (\0 is from chat gpt)
        }
}
```

```
int contains_values(double r_one[])
{
        int num_values = 0;

        for (int i = 0; i < 11; i++)
        {
                if (r_one[i] > 0)
                {
                        num_values++;
                }
        }

        if (num_values > 0)
        {
                return (1);
        }
        else
        {
                return (0);
        }
}
```

Conclusion

Successes

My biggest success is that the program worked! The user could input their needs, and the program outputs possible solutions. It saved time, and all results are accurate.

Another major success is that my program is practical. Many times in class we have utilized 555 Timers, and having a calculator like this would have made things much easier. I may hang on to this program to use in the future for that very reason.

Problems

One problem I encountered was user input. When the user entered something wrong (like letters instead of numbers), the scanf function would store all the extra characters the user entered, leading to multiple error messages. Also, when the user would push enter, the scanf function would store that entry for the next prompt. This caused many unnecessary error messages. After researching the issue, I found a line of code that would clear the input in the scanf function using getchar(), and how to correctly use the scanf function to ignore the "enter" as input.

Another difficulty was in the calculations. Although the math is fairly simple, it was hard to keep track of many different units in the code. Also, if the duty cycle was 50% or less, a diode and different math would be required for the circuit. It took me a while to figure out how to best accomplish this.

Improvements

One improvement I could make is simplifying my input functions. The "get user input" states/functions pretty much do the same thing: prompt the user for input, check if it's valid, and return the value. I wonder if it is possible to create one reusable function for each of these states.

Another improvement I would make, if there was more time, is printing a circuit diagram for the user. The user could select the row number that best suits their needs, and the program would print a diagram with the values. All in all though I learned a lot from the project, and I am proud of my program.