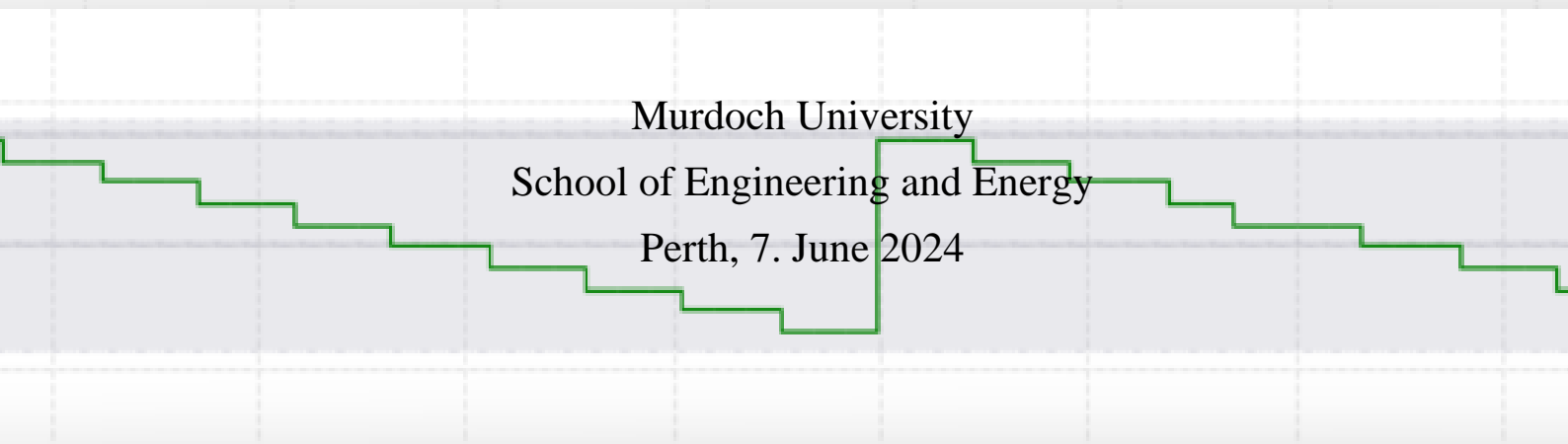
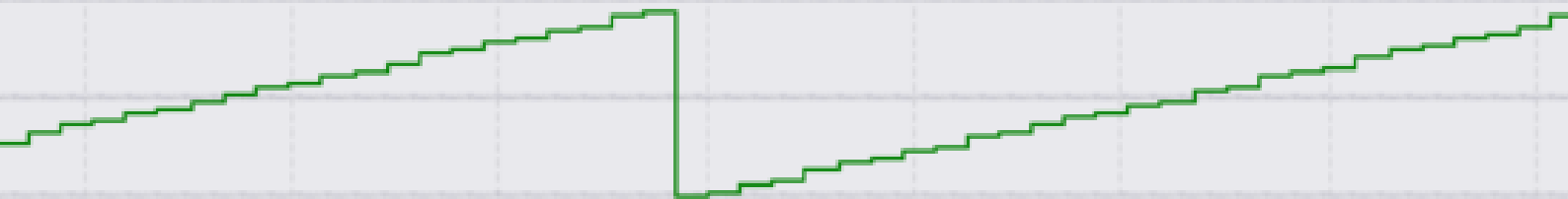
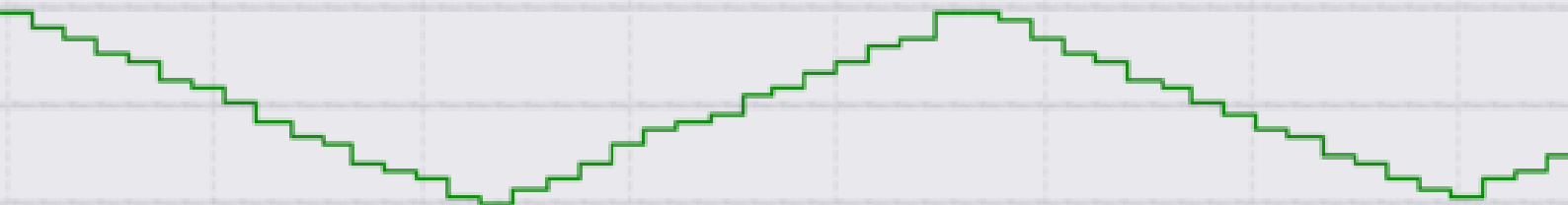




ENG251 – PLC Systems

Spring Semester 2024

Project 3 – Function Generator



Murdoch University
School of Engineering and Energy
Perth, 7. June 2024

Group Maverick

Gert van der Merwe

Electrical Engineering

gert@mavericsystems.com.au

Merlin Blickisdorf

Electrical Engineering

merlin.blickisdorf@stud.hslu.ch

Project 3 – Function Generator

Automation with the Siemens PLC

Supervisors:

Dr. Sheikh Azid

Nicholas O'keeffe

Murdoch University – School of Engineering and Energy

Perth, 7. June 2024

Table of Contents

Abstract	3
1 Theory	4
2 Objective	4
3 Function Generator.....	5
3.1 Approach	5
3.2 Technical Information	5
3.3 TIA Programming	6
3.3.1 Sine Wave and OB35 Cyclic Interrupt.....	6
3.3.2 Square Wave	7
3.3.3 Triangle Wave	8
3.3.4 Sawtooth Wave	8
3.3.5 Initialize Custom Wave	9
3.3.6 Custom Wave	11
3.3.7 OB1 Main and OB100 Reset.....	13
3.4 Simulation	15
3.5 Circuit and Testing	19
3.6 Result and Analysis	22
4 Conclusion.....	23
5 Outlook.....	23
6 Directories	24
6.1 References	24
6.2 List of Abbreviations.....	24
6.3 List of Tables.....	24
6.4 Table of Figures	25
7 Appendix	26

Abstract

In the unit «ENG251 – PLC Systems» from Murdoch University Perth a third and final task has been given. This project, as well as the first one, could be defined by the team. However, there are some requirements that must be fulfilled, for example there must be an analogue input and output to be used in the project. Moreover, the project must conclude a higher complexity than the first one.

As in the first and second project, the heart of the third project lies in the PLC, short for “Programmable Logic Controller”. For this project, the Siemens S7-300 series PLC will be used. It has inputs and outputs that can make use of several sensors, like switches, push buttons or potentiometers and actuators like lamps, motors, cylinders, liquid, or gas pumps and suction cups.

The unit provides fundamental knowledge in PLC Systems. Furthermore, it teaches their students to independently solve automation problems and learn new technologies and methods like ladder logic LAD, STL, FBD, timers, counters, simulators, digital and analog I/O, PID control, state machines and many more.

The team that wrote the following report, consisting of Gert van der Merwe and Merlin Blickisdorf, has received the task to write a program for the Siemens PLC to solve the team defined task of building a function generator with the Siemens PLC. To be more precise, the amplitude and frequency must be controllable by a potentiometer during the use.

This report includes how the team approached the task, how it solved the upcoming issue, the results, and some conclusions on how to improve in future projects.

1 Theory

A “Programmable Logic Controller”, or for short PLC, is widely used in the industry due to its versatile use case. PLCs help to solve the control of complex problems while being rather simple to program and maintain.

In this unit the Siemens S7-300 is being used, the exact model can be seen in Figure 1. Its model number is as follows: CPU 314C-2 DP. This is a Siemens PLC of the middle-range of their lineup. The lineup of Siemens consists of the following models: Low-range S7-200, middle-range S7-300 and upper-range S7-400 or higher.

The PLCs from Siemens normally possess digital and analog inputs and outputs which can be used to control a wide range of sensors and actuators. The heart of the PLC consists of a microcontroller which has several key technologies like timers, counters, analog to digital and digital to analog converters, digital storage and many more. As a matter of fact, the Siemens PLCs can be extended if the provided inputs and outputs are not enough. Furthermore, they are conceptualized to withstand the harsh environment of the industry, which can be high temperatures, strong vibrations, high humidity, or high electrical noise. PLCs generally can be programmed with simple ladder logic, which is being sequentially processed by the microcontroller inside of it.¹

PLC System are being used all over the industry. For example, in the robotic, paper and steel, automobile, glass, or food processing industry. Another fact is, that they also find use on independent places, like on small and big ships or even on planes.

The PLCs from Siemens are being programmed either in STL, LAD or FBD in a software called “Totally Integrated Automation”, or for short TIA.



Figure 1: Siemens S7-300 PLC

2 Objective

The task for the third project of the unit «ENG251 – PLC Systems» has been defined by the team itself. It consists of building a simple function generator with the Siemens PLC. The idea behind it is to create four general waves, namely a sine-, square-, triangle- and a saw-wave. Additionally, there should be a custom wave that should be configurable within ten steps while using a potentiometer to define them. Finally, while in action the amplitude and frequency must be manipulatable with two different potentiometers. If the system is idle, a red LED should blink, if the configuration mode is on a green LED should blink and if a wave is being outputted, the green LED should stay on continuously.

¹ (Siemens, 2024)

3 Function Generator

This chapter holds the main information about this project like how the team approached the task, the technical information on how the automation works, the programming of the PLC itself and the used circuitry and testing of said circuitry. Furthermore, the results are being presented and evaluated.

3.1 Approach

The approach to this project has been defined by the team prior to the start. It has been decided that both members of the team do the programming independently and that those two programs should be compared afterwards. This way both students have the highest possible learning curve, and the best result can be evaluated. Furthermore, good ideas can be merged if one of the solutions is better but lacks a key function that the other possess.

3.2 Technical Information

The goal of this automation project is to build a function generator using the Siemens PLC. To reach that goal, several technical details had to be defined before the project started. First, the analogue and digital inputs and outputs needed to be set. And secondly, the rough functionality of the function generator needed to be defined.

In the following Table 1 all inputs and outputs of the system can be seen. Moreover, the type of the in- and outputs can be seen, if it is analogue or digital and the in- output name in the system of the PLC. Additionally, their purpose can also be seen.

Table 1: Inputs and Outputs of the PLC

Input	Output	Analog	Digital	Name in PLC	Purpose
x		x		IW752	Potentiometer Amplitude
x		x		IW754	Potentiometer Frequency
	x	x		QW752	Oscilloscope (Visualization)
x			x	I124.0	Push-button Start/Stop Sine-Wave
x			x	I124.1	Push-button Start/Stop Square-Wave
x			x	I124.2	Push-button Start/Stop Triangle-Wave
x			x	I124.3	Push-button Start/Stop Saw-Wave
x			x	I124.4	Push-button Configure Custom-Wave
x			x	I124.5	Push-button Start/Stop Custom-Wave
	x		x	Q124.0	LED green
	x		x	Q124.1	LED red

The functionality of the function generator has been defined as follows: There are two potentiometers with whom the amplitude and the frequency can be manipulated. Moreover, there is a analogue output, that should be connected to an oscilloscope which shows the generated waves of the PLC. Furthermore, there are six digital inputs in form of push buttons for the different waves and two digital outputs, one for a green LED and one for a red LED.

3.3 TIA Programming

The Siemens PLC gets programmed either with “Statement List Programming” (in short STL), “Ladder Diagram” (in short LAD) or “Function Block Diagram” (in short FBD), in a software called “Totally Integrated Automation” (in short TIA).

In the following chapters, the TIA programming of the function generator is being described. Each wave got programmed inside a function call and the initializing of the custom wave got programmed inside a function block because it uses a global variable to save the voltage levels. Furthermore, the OB1 main program, the OB35 cyclic interrupt and the OB100 reset get explained.

3.3.1 Sine Wave and OB35 Cyclic Interrupt

The base sine wave is being created using the sine function of the TIA environment. The code for it is directly inside the OB35 cyclic interrupt. In the function call two “Sine-Wave Generator” the base sine wave gets unscaled and copied to the output. But that function call is practically empty except for this unscale function. This has been done because the base sine wave is being used in every other wave function, and therefore must run all the time another wave runs. The sine wave is basically being built using the following sine function:

$$v(t) = A * \sin(w * t)$$

Where “A” is the amplitude which gets created using the data from the amplitude potentiometer and “w” being the angular frequency which gets calculated using the data from the frequency potentiometer and multiplying this value by two times pi. To capture the time “t” a counter in a global variable gets incremented and multiplied by 0.1 each time the program runs through it, since the OB35 cyclic interrupt gets polled each 100 ms.

If another wave gets picked, 0.0 is being written into the output. That way, each time a wave starts it starts at the ground value of 0 V. Additionally, the base sine wave only gets generated while one of the waves is active, thereby saving energy when the system is idle. This is being done AND-ing all rungs correlating with the base sine wave with an activate-deactivate bool variable that is being set true if one of the waves is active.

The problem with the potentiometer analogue input is that the potentiometers are not precise and have a lot of jitter and noise. To reduce the negative side effects of that phenomenon an averaging function has been put in place. This function basically adds up the value of the potentiometer input until a certain threshold is reached, where it divides the added-up values by the number of counts. The maximum number of counts is predefined in a static variable for ease of change. To count the add-ups a global variable has been put in place. This variable gets set to zero if the value inside the static variable gets reached, see Figure 2. The averaging function itself (or a part of it) can be seen in the Figure 3. Behind the out most right part of the figure is the division of the “countAmpl” (holds the value of the added-up amplitudes) by the number of counts.

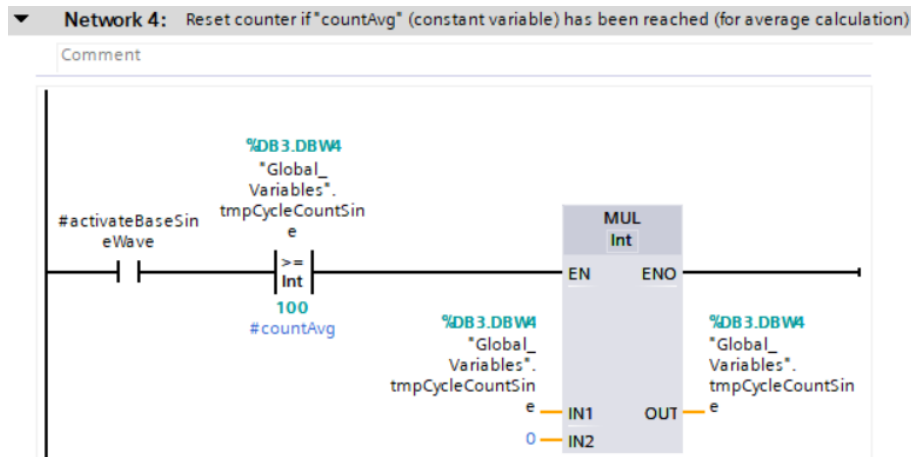


Figure 2: Reset of the averaging function

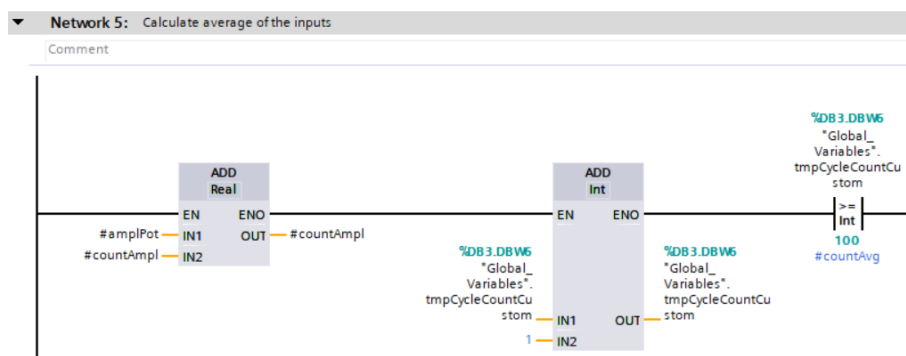


Figure 3: Averaging function

3.3.2 Square Wave

The square wave is straightforward and gets built out of the sine wave. If the sine wave is positive a positive square gets copied into the output and the other way around in the negative scenario. In the following Figure 4 a cut out of that program can be seen.

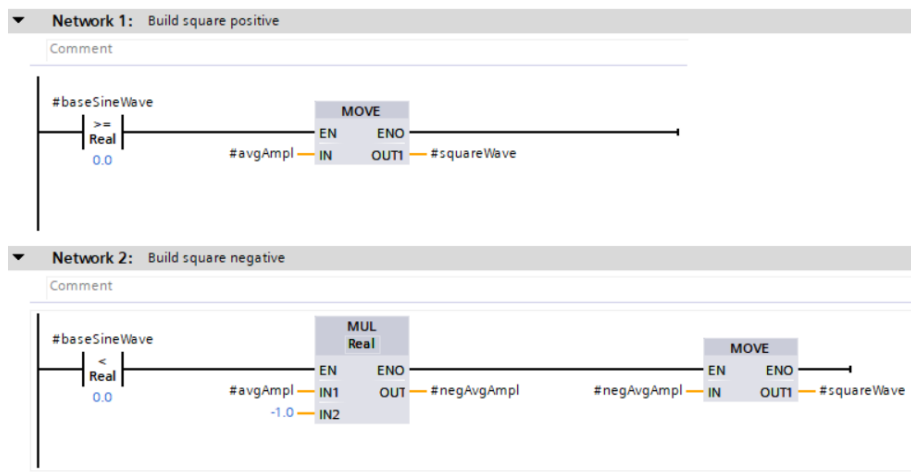


Figure 4: TIA program square wave

3.3.3 Triangle Wave

The triangle wave is being built counting up a value and writing this value with the correct scaling into the output, if the value reaches its highest point, it starts counting down again resulting in a pyramid form. This function is based on the square wave that has been mentioned before. This is due to the needed correct pacing of the wave and the zero crossings from the frequency of the sine wave. In Figure 5 the counter for going up and creating a stairway can be seen. Each time the program runs through the impulse tag gets triggered which increments the counter. This happens so long as the square wave is either positive or negative. If the square wave changes polarity this counter gets reset and the second counter (which is a count-down counter) receives the value from the first and decrements this value until the whole thing begins anew.

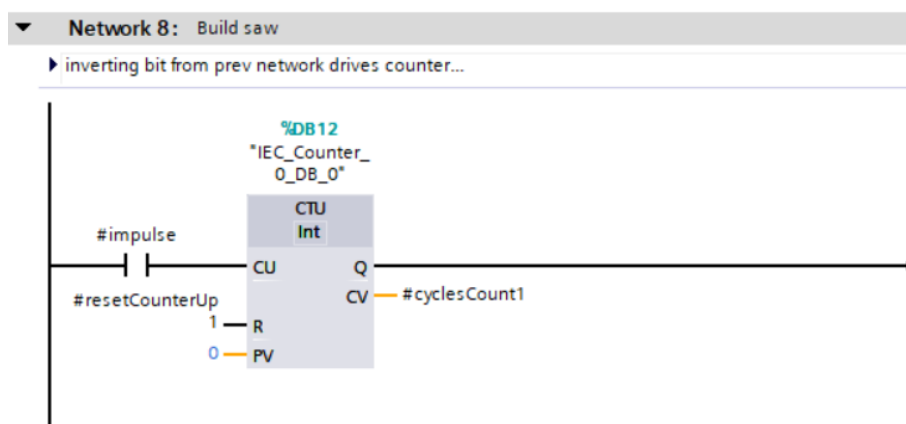


Figure 5: TIA program triangle wave: Counter

Figure 6 shows how the wave gets created. First the count value or gets moved into the wave value and secondly the wave gets divided by the steps evaluated at start that determine where the voltage level is in the period and then the wave gets multiplied with the correct average of the potentiometer's amplitude. Finally, it gets scaled to the correct value.

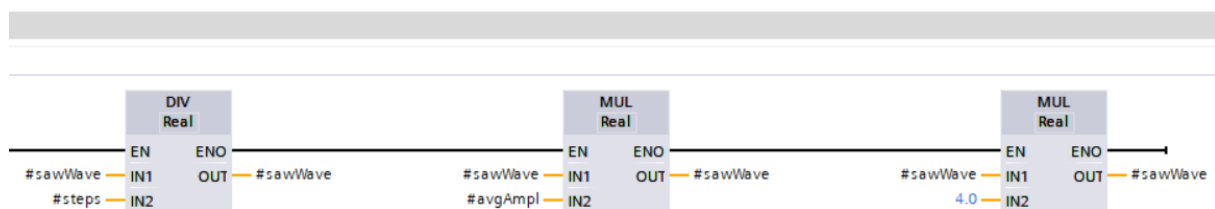


Figure 6: TIA program triangle wave: Creation of the wave

3.3.4 Sawtooth Wave

The saw tooth wave is more or less the same as the triangle wave in exception that it does not count down, but it rather resets the counter and goes to zero as soon as it reaches the peak value or highest point. Therefore, it will not be further explained here.

3.3.5 Initialize Custom Wave

To generate a custom wave, the custom voltage levels must first be defined. Therefore, a function block is due that asks the user for these custom voltage levels to store them for later use. This function block does exactly that. The custom wave is split up into ten individual voltage levels, that need to be determined by the user before the custom wave starts.

At first, the function directly outputs the current voltage amplitude level of the potentiometer to the oscilloscope, so the user can see where the voltage level is at. Then the user can press a push button to save the current voltage level. After that, the next voltage level can be set with the potentiometer and again saved with a press of the push button. This goes on until the push button has been pressed ten times after which the custom wave starts immediately.

To realize this endeavor, a state machine has been created, see Figure 7. It can be seen that every time a counter increments, the state changes from one to another. In every state, the voltage level of the potentiometer gets saved inside a global variable (of the function block) continuously until there is a transition to a new state. Or differently described, the voltage level does not get written once it gets into the state but rather overwritten all the time until there is a transition to the next state. Thereby, ensuring the last and correct voltage value of the potentiometer gets saved into the global variable of the function block. When the counter hits ten, the function call will exit since this is the exit condition of the state machine. This is to ensure a smooth transition from the initializing function block to the running of the custom wave function call.

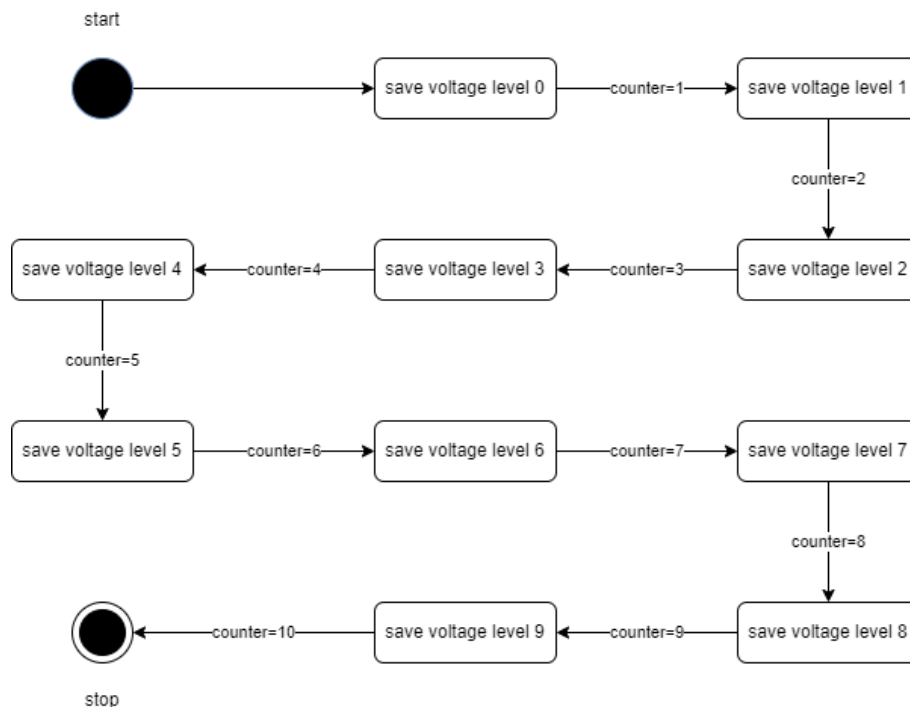


Figure 7: Initialize Custom Wave: State machine

In the following Figure 8 and Figure 9 the implementation of the first state can be seen. As mentioned before, the condition of the transition is the incrementation of the counter, since the counter starts with zero the first state will also be in state 0 at first. Additionally, the state 0 gets also initialized in the OB100, but more on that later.

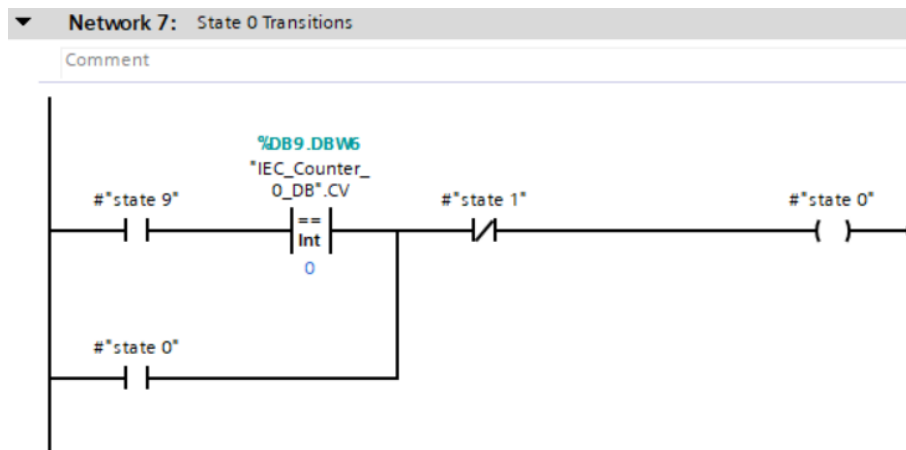


Figure 8: Initialize Custom Wave: State 0 Transactions

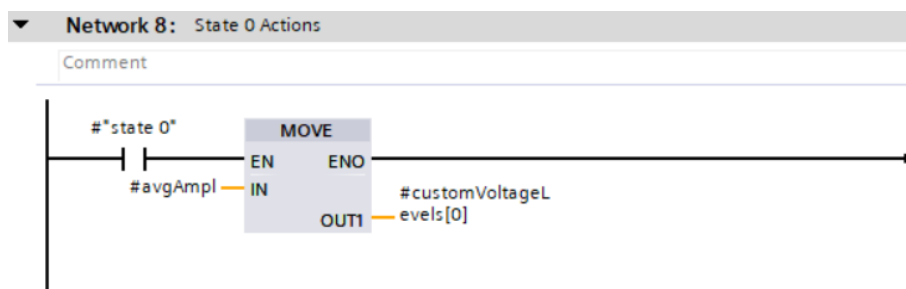


Figure 9: Initialize Custom Wave: State 0 Actions

The Figure 9 shows the continuous saving of the current voltage level of the potentiometer saved into the variable “avgAmpl”. This value as in the sine wave uses the same averaging function to reduce the jitter and noise of the potentiometer.

In the main program, see Figure 10, the initialization tag for the custom wave gets reset after leaving the custom wave function call. Thereby ensuring that the initialization must be done again after entering the custom wave function for a second time. This could be removed if the initialization should only be done once at the first start of the custom wave.

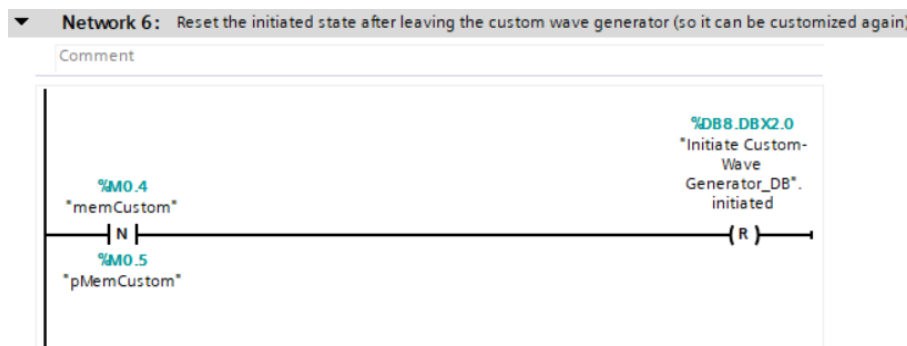


Figure 10: Initialize Custom Wave: Reset the custom wave initialization tag

3.3.6 Custom Wave

To create the custom wave, several issues had to be overcome. But, as before the base of this wave is the sine wave. The sine wave gets used to determine when each part of the ten different voltage levels must be turned on. In the Figure 11 a sketch of a sine wave can be seen, which got split into ten steps. If the current sine value is between step zero and step 1, the stored voltage level would be copied to the output and correspondingly for all other ranges. The problem with that was, that if there is an if condition comparing the two voltage levels between step one and step two then there are two possible results, since the sine wave is not linear in one period. Linear functions only have one possible result for each y-axis point. But this is not the case with a sine function. However, if the sine function gets split into four regions: a positive rising edge, positive falling edge, negative falling edge and a negative rising edge, then each of these four regions could be regarded as linear. Thereby, the position of the current sine value can be determined precisely. Furthermore, only the first two voltage level steps, step one and two, must be determined, since all other steps are a copy or a negation of these steps. All the above-mentioned informations have been put into the sketch to visualize the chosen approach.

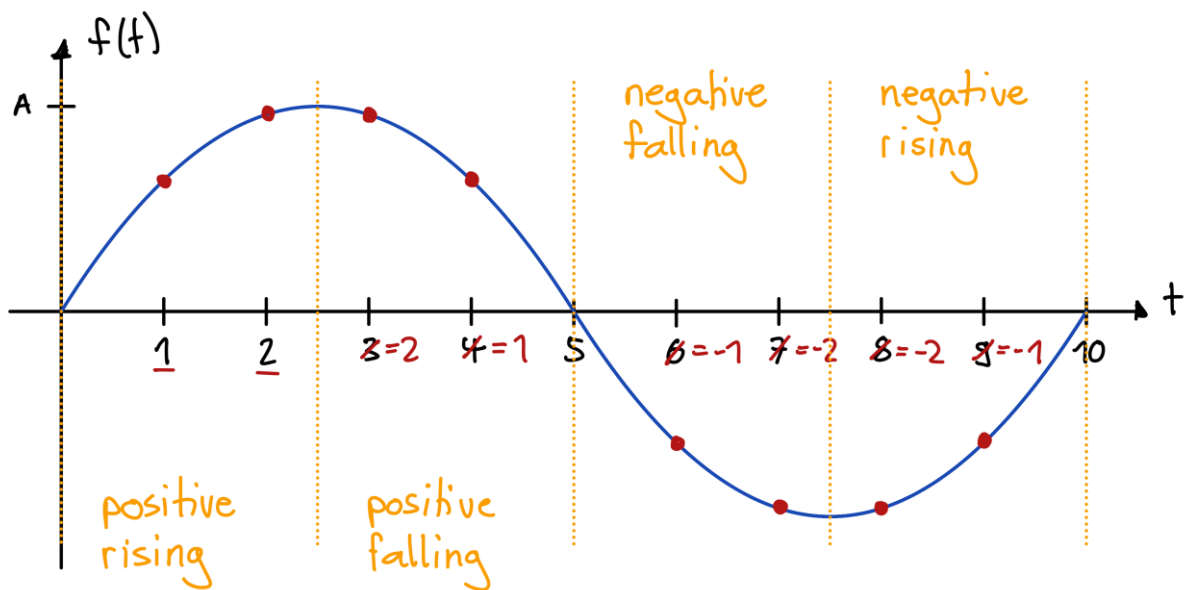


Figure 11: Custom Wave: Splitting up the sine wave

The voltage levels of step one and two can be calculated as follows: (example for an amplitude of 10 V and a frequency of 1 Hz)

$$V_{t=step1} = A * \sin(\omega * t) = 10 \text{ V} * \sin(2\pi * 1 \text{ Hz} * 0.1 \text{ s}) \approx 5.88 \text{ V}$$

$$V_{t=step2} = 10 \text{ V} * \sin(2\pi * 1 \text{ Hz} * 0.2 \text{ s}) \approx 9.51 \text{ V}$$

Concluding the formula the two important voltage levels for a sine wave with the amplitude of 10 V and a frequency of 1 Hz (the frequency is irrelevant for this calculation) are 5.88 V for the first step and 9.51 V for the second. With those two voltage levels and the split up in four sine parts the current position of the sine wave can be located uniquely.

In the following Table 2 the voltage ranges for determining the current position of the sine wave can be seen.

Table 2: Voltage ranges of the sine steps

Step range	Voltage level (if A = 10 V) [V]
0 <= sin AND sin < 1	0.0 <= sin AND sin <= 5.88
1 <= sin AND sin < 2	5.88 <= sin AND sin < 9.51
sin >= 2	sin >= 9.51
2 >= sin AND sin > 1	9.51 >= sin AND sin > 5.88
1 >= sin AND sin > 0	5.88 >= sin AND sin > 0.0
0 >= sin AND sin > -1	0.0 >= sin AND sin > -5.88
-1 >= sin AND sin > -2	-5.88 >= sin AND sin > -9.51
sin <= -2	sin <= -9.51
-2 <= sin AND sin < -1	-9.51 <= sin AND sin < -5.88
-1 <= sin AND sin < 0	-5.88 <= sin AND sin < 0.0

In the Figure 12 the rising or falling edge of the sine wave gets determined, with a simple calculation, as in the following formula: (the division by two is not programmed, since it is not necessary, the only thing of interest is if the number is positive or negative)

$$V_{RisingOrFalling} = \frac{V_2 - V_1}{2}$$

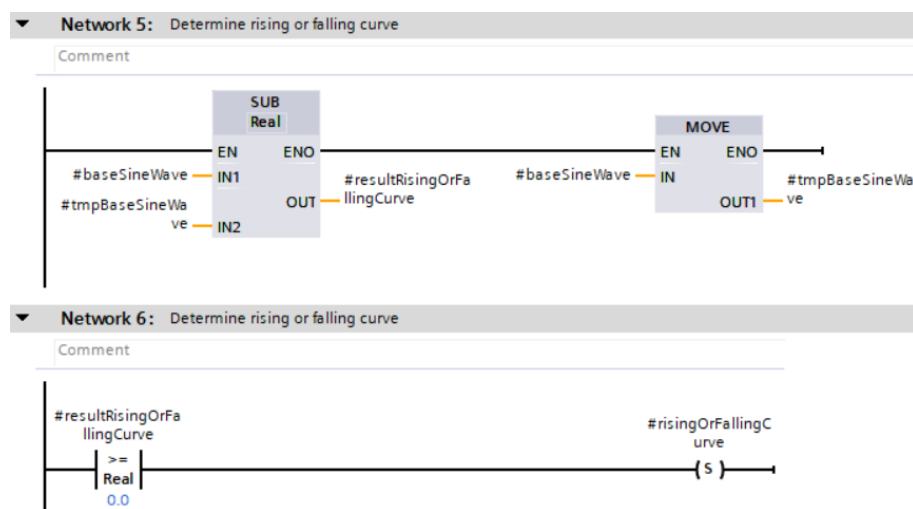


Figure 12: Custom Wave: Determine rising or falling edge

The next Figure 13 shows the calculation of the voltage level of step one and four.

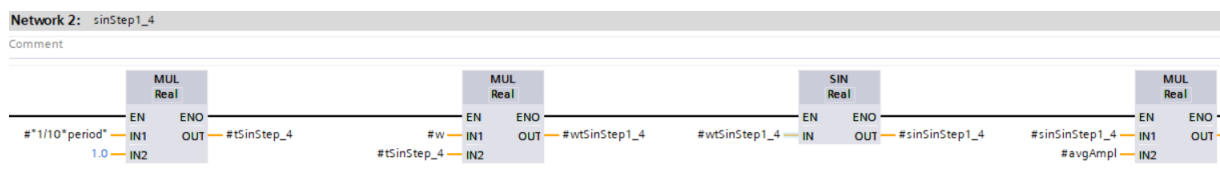


Figure 13: Custom Wave: Calculation of step 1 and 4

Finally, Figure 14 shows the range of the first step (0 V to the first step) which determines when the sine wave is in this range and copies the saved value from the global variable into the output if it is true.

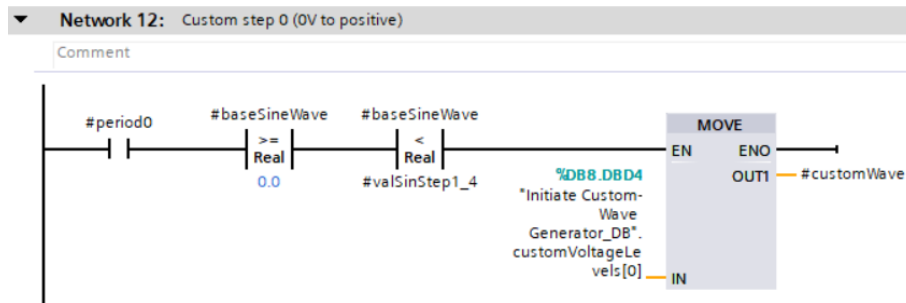


Figure 14: Custom Wave: Determine location of the sine wave

3.3.7 OB1 Main and OB100 Reset

Finally, the main OB1 function. This function incorporates all the other function calls from the different wave forms and the function block of the initialization for the custom wave. To ensure that only one wave is active at a time, all the inputs (or memory tags that save if a wave is on or not) are AND-ed with the input of the push button, see Figure 15. To use push buttons for turning the waves on and off, a toggle function block is being used. The same function block also has outputs for the LEDs to reset them if a wave gets reset to ensure that an LED does not unintentionally shine.

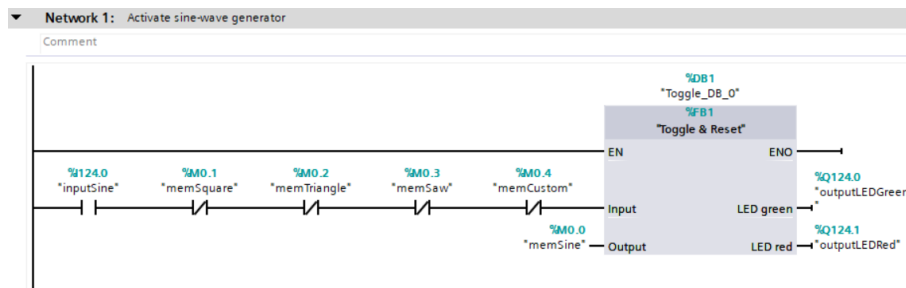


Figure 15: OB1: Activate sine wave generator

If the program is in idle mode (no wave at the output), the red LED blinks, this can be seen in the Figure 16 below. Similarly, the green LED must blink when the initializing of the custom wave is being done. If a wave is being outputted, the green LED shines constantly.

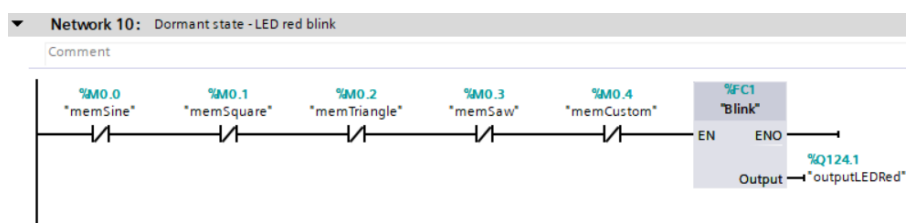


Figure 16: OB1: Dormant state LED red blinks

Figure 17 below shows the reset of the real time since start and the cycle count in the OB100 reset function. This function runs only once when the PLC restarts.

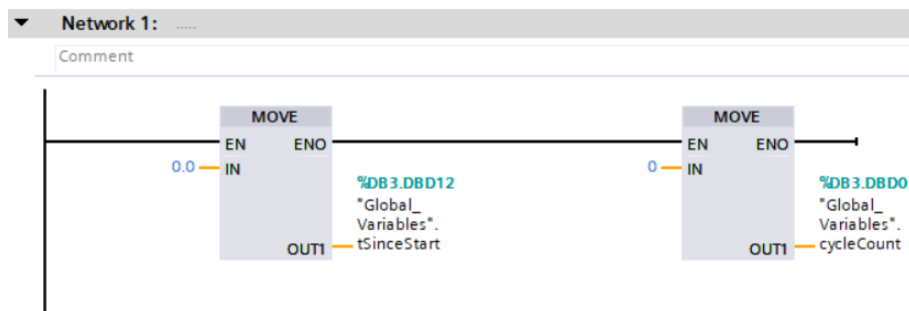


Figure 17: OB100: Reset time since start and cycle count

Similarly, the following Figure 18 shows the rest of the state machine state to state 0, when the PLC restarts to initialize the custom wave correctly if this function block is being called.

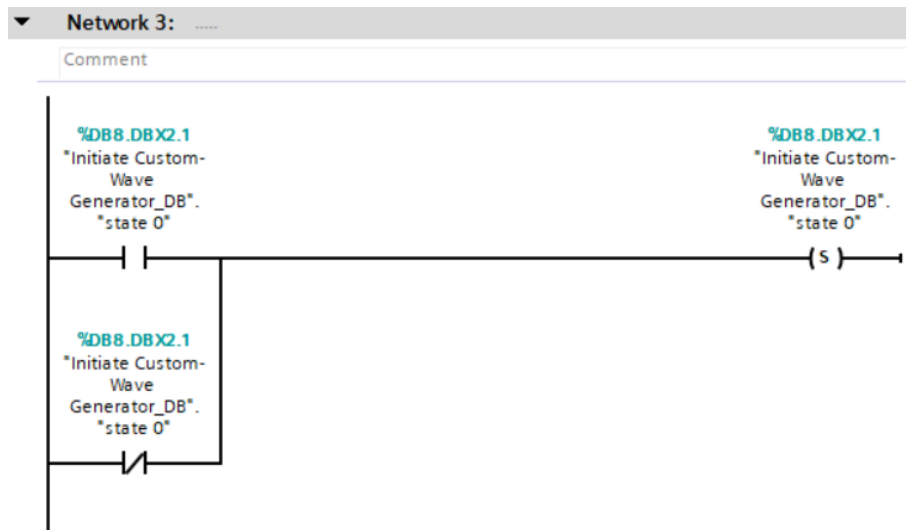


Figure 18: OB100: Reset state of state machine to state 0

3.4 Simulation

Before testing the circuit, the TIA program got simulated thoroughly. At start, the analog input values were incorrect. This error happened due to the input only being 0 to 10 V, instead of the 0 to 12 V of the power source and because the output has values from -10 V to 10 V. To reach 10 V as amplitude the following formula needed to be considered:

$$\begin{aligned} \text{input}_{\text{range}} &= 0 \dots \frac{2^{n_{\text{bits}}}}{2} * V_{\text{usable}} = 0 \dots \frac{2^{16}}{2 * 12 \text{ V}} * 10 \text{ V} = 0 \dots \frac{65'536}{24} * 10 \\ &\approx 0 \dots 27'306.7 \approx 0 \dots 27'306 \end{aligned}$$

Concluding the formula, it can be said that the input range for the analog input for the simulation lies between 0 and 27'306. As shown in Figure 19, the value 27 k has been put in which results in a peak amplitude of roughly 10 V. Thereby, the calculations are correct and can be used to determine the correct input and output value of the PLC. On the other hand, the frequency is not affected by this voltage cut off. However, since the cyclic interrupt polls with 100 ms the highest theoretical frequency is as shown in the following formula:

$$f_{\text{highest}} = \frac{1}{n_{\text{samplingrate}}} * \frac{1}{T} = \frac{1}{5} * \frac{1}{T} = \frac{1}{2 * 0.1 \text{ s}} = 1 \text{ Hz}$$

Followed by that, the highest possible frequency that this function generator can output is just 1 Hz, which is slow compared to function generators used in the industry.²

In the following picture series, Figure 19, Figure 20, Figure 21, Figure 22 and Figure 23 the simulation of all different wave forms can be seen. It proves that the code works as expected under clean circumstances.

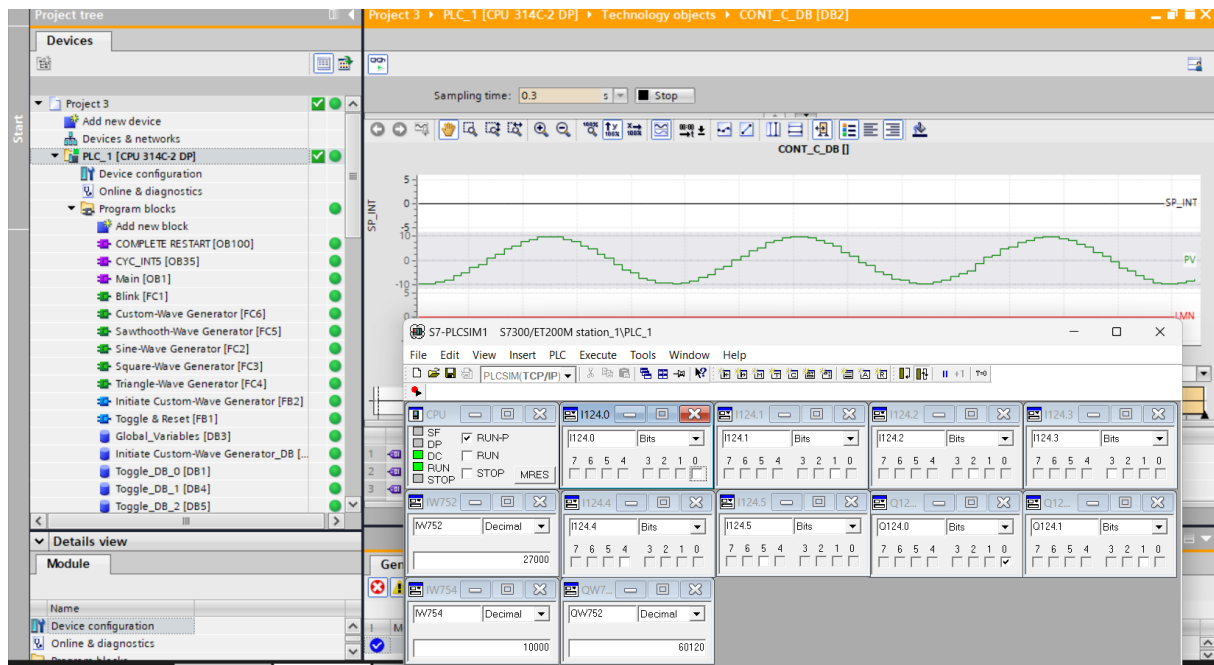


Figure 19: Simulation of the sine wave

² (Okanta, 2021), (Poole, 2024)

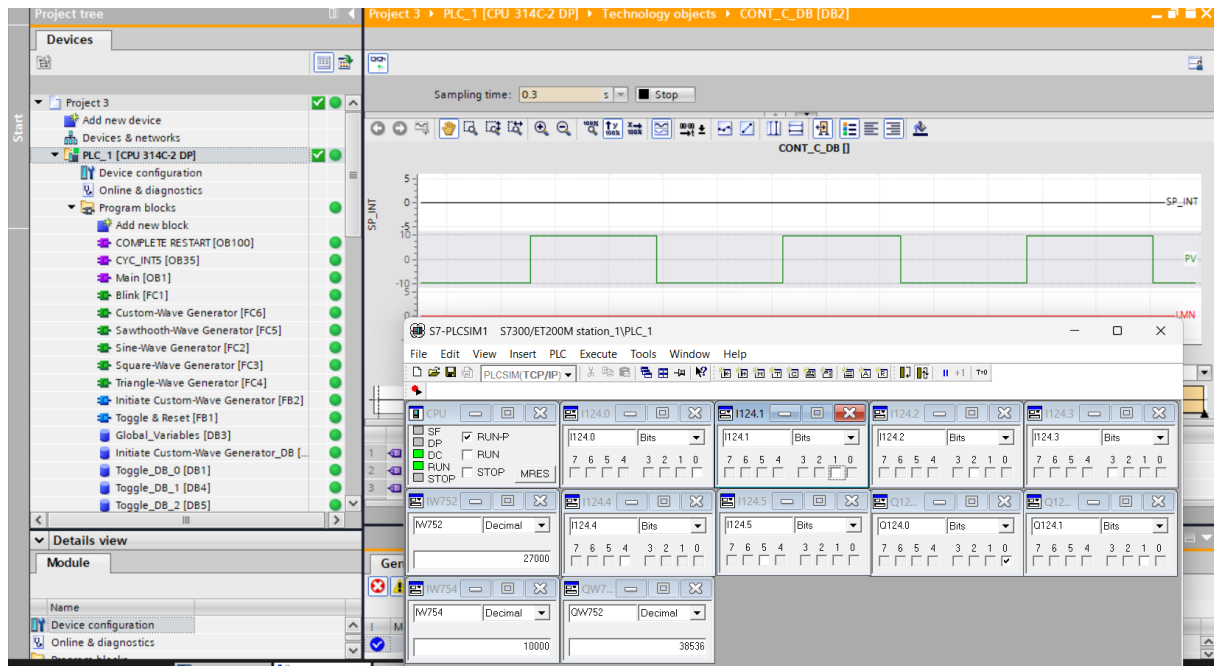


Figure 20: Simulation of the square wave

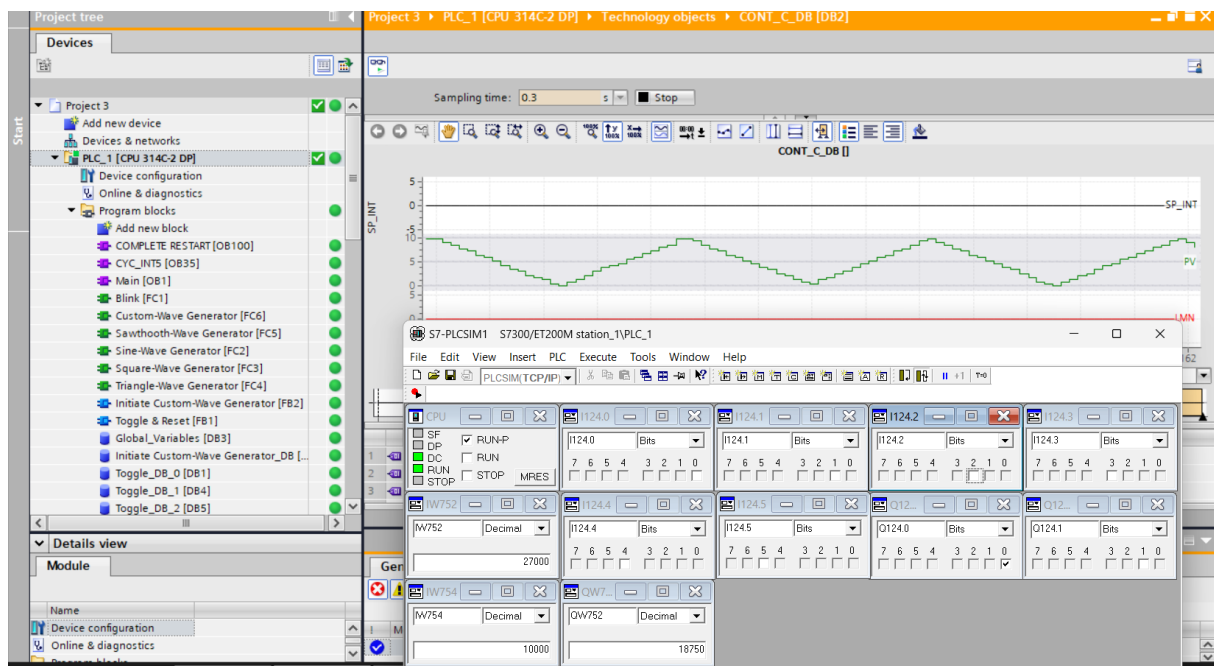
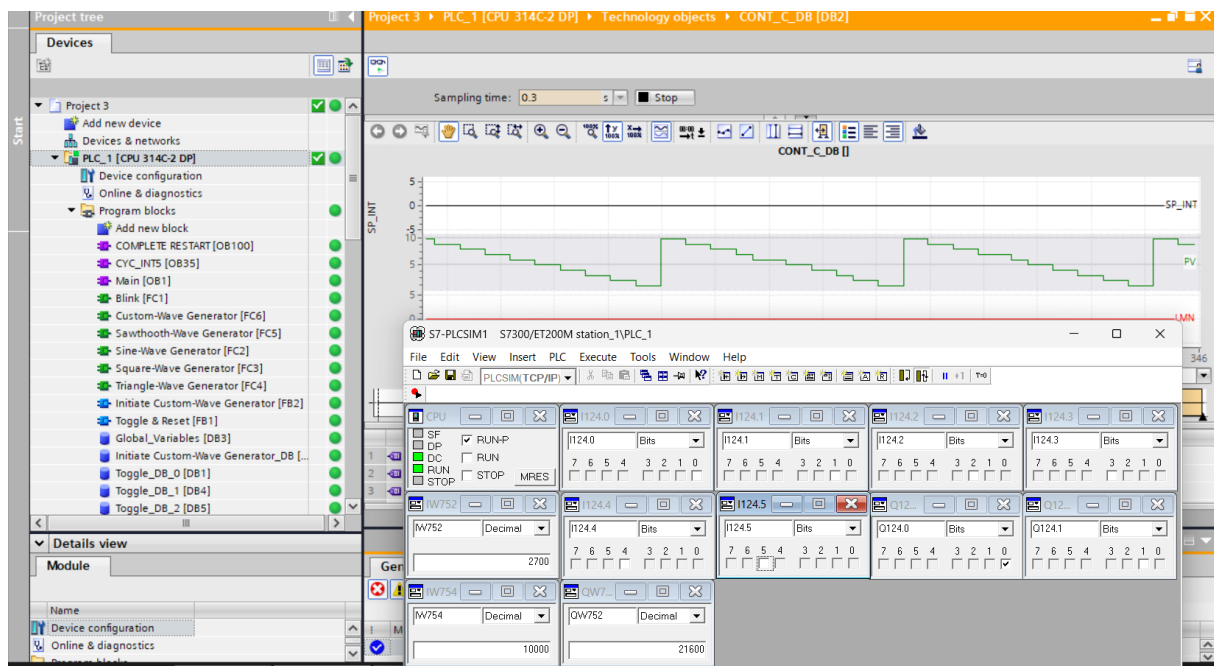
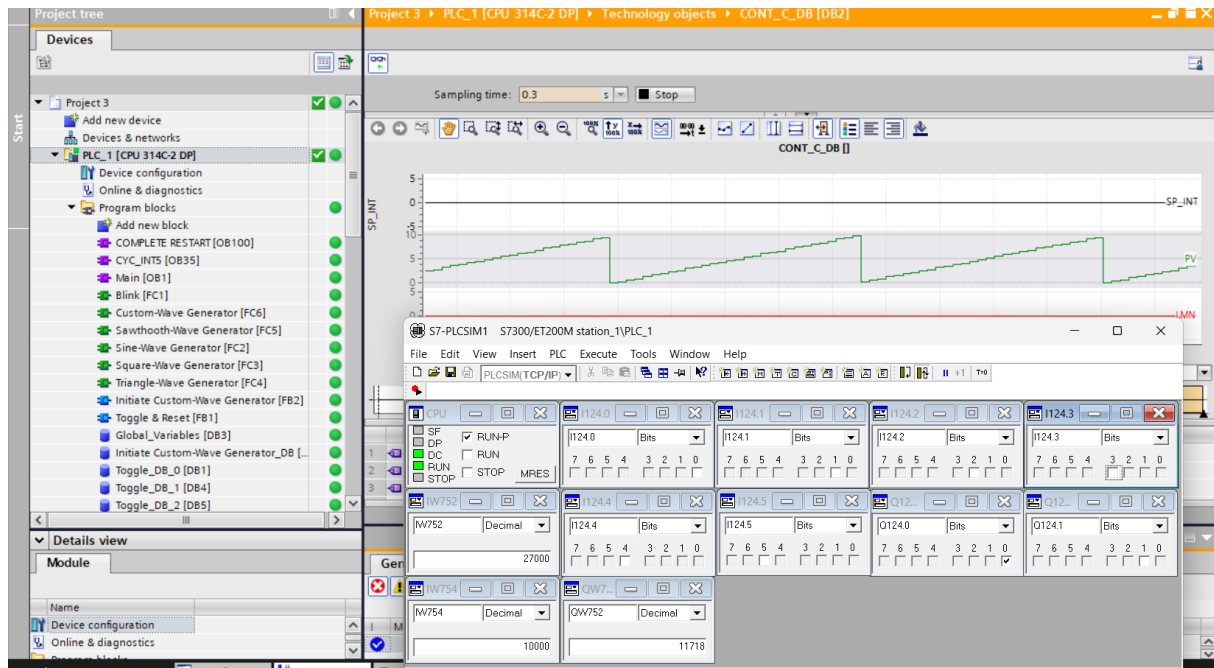


Figure 21: Simulation of the triangle wave



To create the custom wave, the values of the following Table 3 have been used as input. The result is as shown in Figure 23 an inverted saw wave.

Table 3: Voltage steps of the simulated custom wave

Step	Analogue value	Real value [V]
0	27'000	9.77
1	24'300	8.77
2	21'600	7.81
3	18'900	6.84
4	16'200	5.86
5	13'500	4.88
6	10'800	3.9
7	8'100	2.93
8	5'400	1.95
9	2'700	0.98

3.5 Circuit and Testing

In the following Figure 24 the circuit of the function generator can be seen. As mentioned before, two analogue inputs and one analogue output are being used. To provide the system with power a 24 VDC line (in red) and a 12 VDC line (in orange) are being used. The PLC, all push buttons and the LEDs are being sourced with the 24 VDC line. The potentiometer however only receives 12 VDC since the analogue input of the PLC only allows voltages in the range of 0 to 10 V.

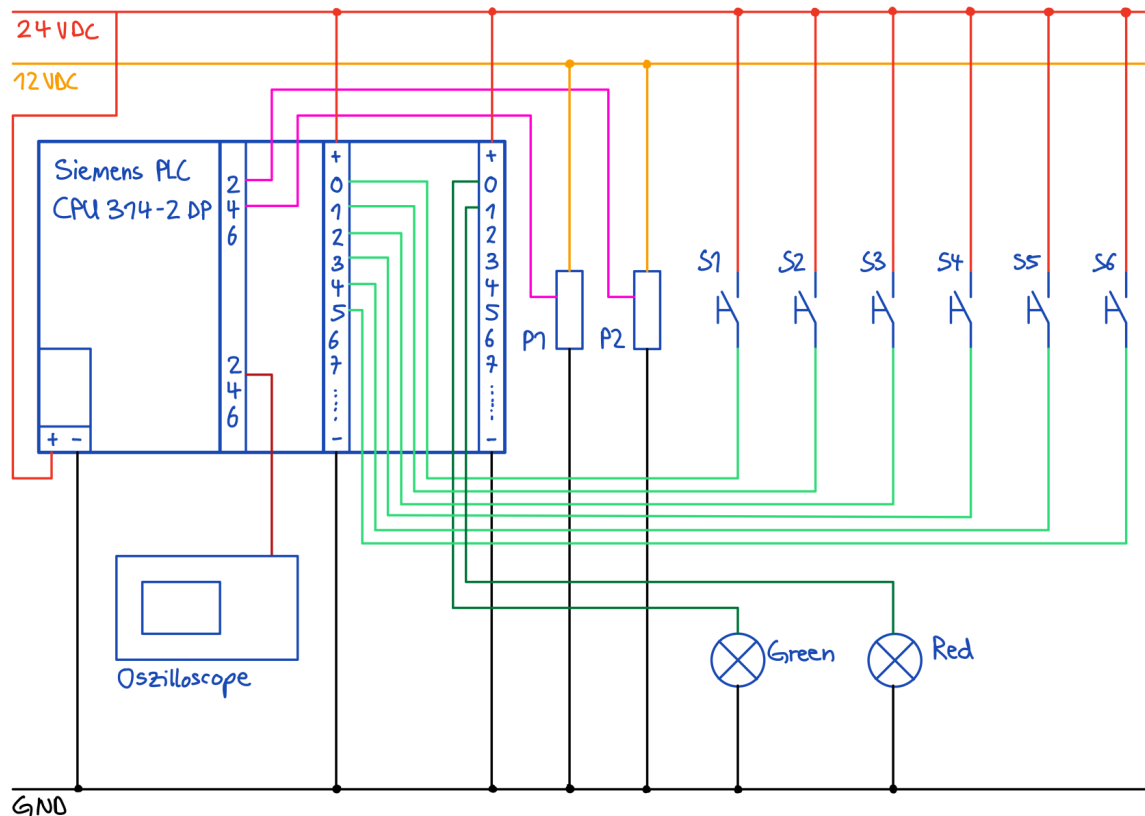


Figure 24: Circuit of the function generator

The numbers showing the connections with the switches and LEDs correspond to the Table 1 which holds the information regarding the inputs and outputs of the PLC. The connection with the oscilloscope can also be seen. It relates to the analogue output Q752 of the Siemens PLC.

In the following Figure 25 a clean build-up of the circuit can be seen. The different components are marked with numbers and named in a legend which is on the bottom right of the figure.

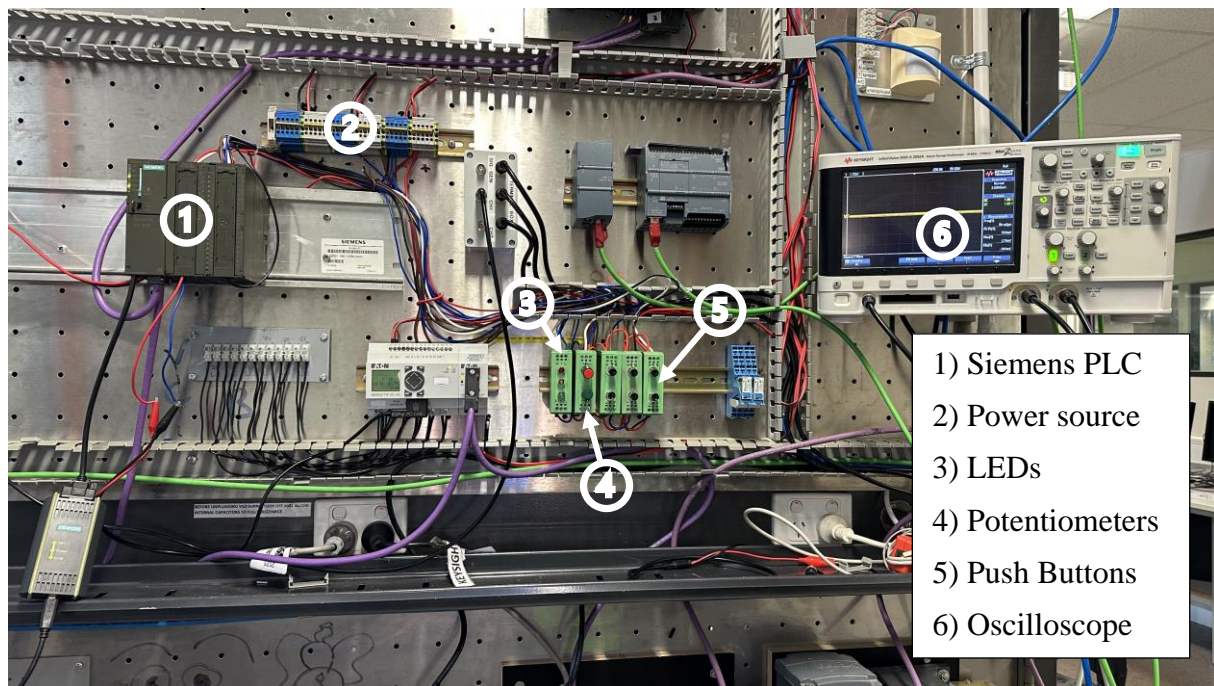


Figure 25: Clean build-up of the circuit

Sadly, while testing the different waves have not been clean all the time, see Figure 26. This is due to the jitter of the potentiometer which are not perfect. This effect could be greatly reduced using the averaging function, that has been evaluated earlier. Still, the effect remains sporadically.

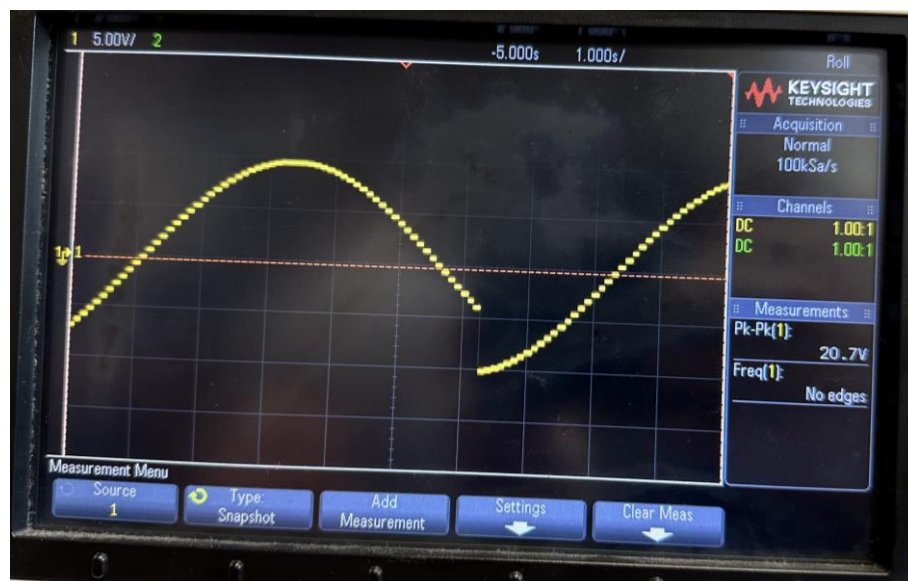


Figure 26: Sine wave with jitter due to potentiometer

In Figure 27 the value of the output is directly connected to the input, through the PLC. It can be seen that there is some jitter that comes from the potentiometer. This jitter lies within a range of ± 0.5 V or about 1.13 V peak to peak, which is significant especially for the frequency control which is quite fragile.

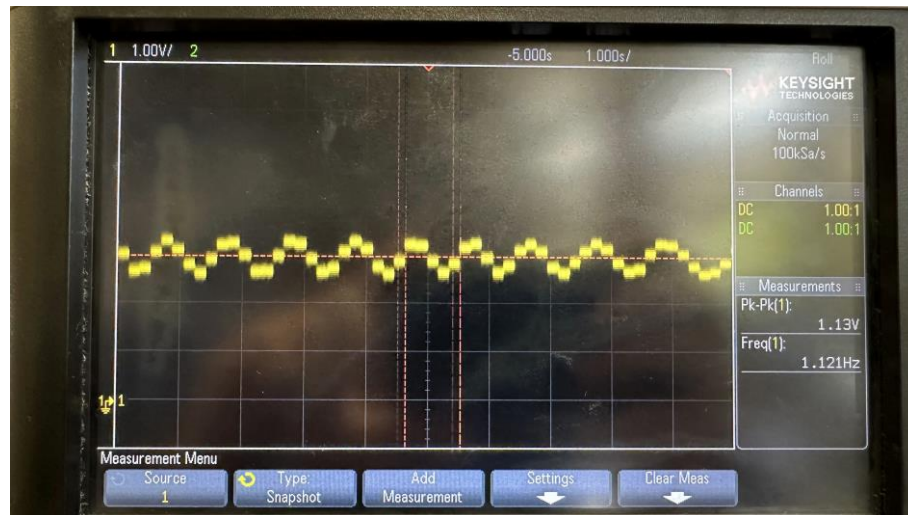


Figure 27: Jitter of the potentiometer in idle state

This phenomenon got recognized while setting the values for the custom wave, where the potentiometer output is directly fed into the input of the oscilloscope to see the value that the user wants to set.

3.6 Result and Analysis

In this chapter, the results of testing the circuit are being presented and evaluated. While programming, there were some issues regarding the toggle function. This was because a function call was used instead of a function block, which saves the old value. After changing that however the function worked perfectly fine.

After putting together the circuit, it has been tested thoroughly. At first, there was no output at all to be seen on the oscilloscope. For the reason being that the scaling of the two potentiometers were off. Luckily, the team thought about this problem while simulating the program. Therefore, this issue could be resolved in no time. After that, all different wave forms were tested. The sine wave worked perfectly fine, although there was the already mentioned jitter that could not be removed. Amplitude and frequency could be changed without concern. However, the reaction time was not great. If there was a change at the potentiometer these changes only followed on the oscilloscope a few seconds later. This is due to the averaging process that takes some time. The square wave also worked flawlessly, which makes sense since it is a simple wave and function. The triangle in contrast had some issues with the jitter. Because of the jitter the wave was not clean and got interrupted on many occasions. Sadly, the saw wave did not work whatsoever. The reason for that could not be found, since it is practically the same function as the triangle it should also work, but it did not. And finally, the custom wave also worked perfectly fine. The custom wave could get initialized as described in a few chapters before, and it showed the correct wave after pressing the set voltage push buttons for ten times. In all the waves the frequency could be manipulated to some extent. If the frequency got increased to high, the PLC was too slow to react upon the signal. For the amplitude it is the same except for the custom wave, whose amplitude cannot be changed on the fly. But that would not make any sense, since the voltage levels got defined prior to running.

The gathered results are legitimate since they could get controlled on the oscilloscope. Furthermore, the results also got presented to the professors who were quite satisfied with the results.

4 Conclusion

In conclusion, it can be said that the project all together was a success. The given task has been fulfilled thoroughly, except for the saw wave which for some reason did not. Additionally, important lessons have been learned which will probably be helpful in the third and last exam of this unit.

At first, it was quite challenging to get a grasp at the simulation of sine waves in the TIA software environment. It took quite some time to get used to the simulation method of the PID-block, especially because of the frequencies and the correct scaling of said frequency. But eventually, everything worked perfectly fine, as can be seen on the cover sheet of this report, since the curves that can be seen on, there are screenshots from the simulations.

Finally, the team of this report is satisfied with these results, especially with the custom wave which was quite satisfying to see since much effort was put into that function.

5 Outlook

If this project was taken any further, the following points must be considered. First, let us talk about optimization. Since there are several function calls in this program minor changes could be made. For example, the averaging of the inputs could be removed from the OB35 cyclic interrupt and moved into its function call. That way the whole averaging function must not be rewritten in the initialization function block, which is now the case. Furthermore, the triangle and saw function could be combined by removing the saw function call and changing the triangle one with adding an option to cut off the count-down of the values. That way the same result of as in the saw function could be reached. Moreover, a low pass filter should be added to potentiometer inputs to reduce the jitter even more.

Finally, it can be said that realizing a function generator with a PLC does not make sense at all. Hence, normal function generators work in a way bigger frequency range and can reach much higher frequencies than just 1 Hz.

But still creating this function generator was fun and very insightful of the workings of the OB35 cyclic interrupt.

6 Directories

This chapter provides all directories like the references, list of abbreviations, list of tables and the table of figures.

6.1 References

- <https://www.globaldata.com>. (2024, March 19). Retrieved from
<https://www.globaldata.com/company-profile/eaton-corporation-plc/>.
- Mughees, N. (2023, March 17).
<https://electronics360.globalspec.com/article/19300/industrial-applications-of-plcs>.
 Retrieved from <https://electronics360.globalspec.com/>.
- Okanta, J. (2021, January 22). www.au.mathworks.com. Retrieved from
<https://au.mathworks.com/matlabcentral/answers/724413-nyquist-shannon-theorem-matlab-code-suffering-from-minor-issues-don-t-know-where-i-went-wrong>
- Poole, I. (2024, June 06). www.electronics-notes.com. Retrieved from
<https://www.electronics-notes.com/articles/test-methods/signal-generators/function-generator.php>

6.2 List of Abbreviations

FBD	<i>Function Block Diagram</i>
LAD.....	<i>Ladder Logic</i>
PID	<i>Proportional Integral Derivate</i>
PLC.....	<i>Programmable Logic Controller</i>
STL.....	<i>Standard Template Library</i>
TIA	<i>Totally Integrated Automation</i>

6.3 List of Tables

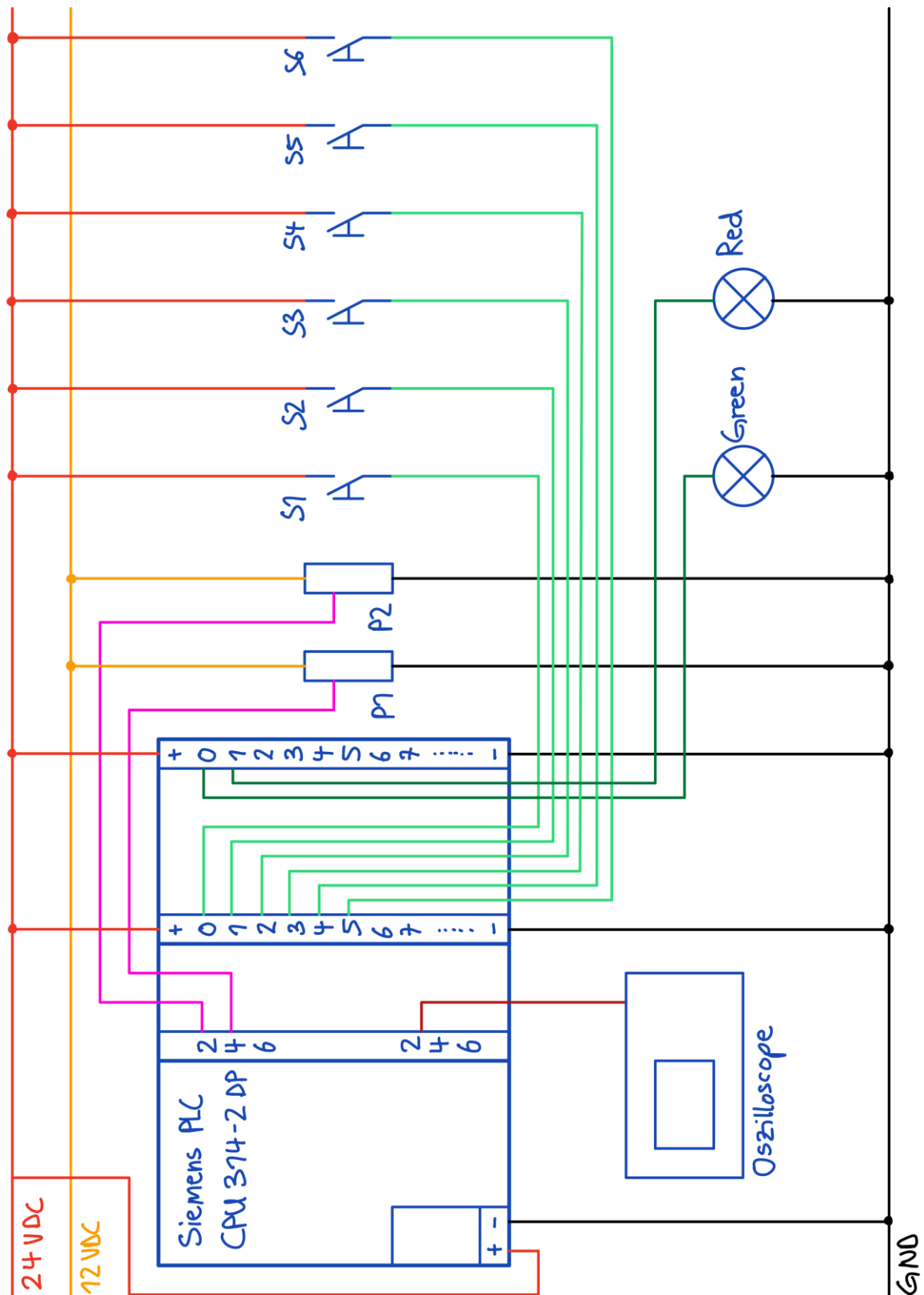
Table 1: Inputs and Outputs of the PLC.....	5
Table 2: Voltage ranges of the sine steps	12
Table 3: Voltage steps of the simulated custom wave	18

6.4 Table of Figures

Figure 1: Siemens S7-300 PLC	4
Figure 2: Reset of the averaging function	7
Figure 3: Averaging function	7
Figure 4: TIA program square wave	7
Figure 5: TIA program triangle wave: Counter.....	8
Figure 6: TIA program triangle wave: Creation of the wave	8
Figure 7: Initialize Custom Wave: State machine.....	9
Figure 8: Initialize Custom Wave: State 0 Transactions.....	10
Figure 9: Initialize Custom Wave: State 0 Actions.....	10
Figure 10: Initialize Custom Wave: Reset the custom wave initialization tag	10
Figure 11: Custom Wave: Splitting up the sine wave.....	11
Figure 12: Custom Wave: Determine rising or falling edge	12
Figure 13: Custom Wave: Calculation of step 1 and 4	12
Figure 14: Custom Wave: Determine location of the sine wave.....	13
Figure 15: OB1: Activate sine wave generator	13
Figure 16: OB1: Dormant state LED red blinks	13
Figure 17: OB100: Reset time since start and cycle count	14
Figure 18: OB100: Reset state of state machine to state 0.....	14
Figure 19: Simulation of the sine wave.....	15
Figure 20: Simulation of the square wave.....	16
Figure 21: Simulation of the triangle wave	16
Figure 22: Simulation of the saw wave	17
Figure 23: Simulation of the custom wave.....	17
Figure 24: Circuit of the function generator.....	19
Figure 25: Clean build-up of the circuit	20
Figure 26: Sine wave with jitter due to potentiometer	20
Figure 27: Jitter of the potentiometer in idle state.....	21

7 Appendix

Circuit



State Machine: Initialize Custom Wave