

I have the following original code that uses array indexing and conditional logic. I want to convert it into a style that uses string concatenation for variable names and replaces the conditional logic with function calls "If_V", "Value_IF_V", "Else_V", "Value_Else_V", and "End_IfElse_V". Here is the code:

Input Code:

```
element_base = "temp_"
```

```
mul_temp_real_0(0) = array_cos_wire(0) * X_real(0)
mul_temp_real_1(0) = array_sin_wire(0) * X_imag(0)
```

```
mul_temp_real(0) = mul_temp_real_0(0) -
mul_temp_real_1(0)
```

```
temp_real(0) = 0 + mul_temp_real(0)
```

```
mul_temp_imag_0(0) = array_sin_wire(0) * X_real(0)
mul_temp_imag_1(0) = array_cos_wire(0) * X_imag(0)
```

```
mul_temp_imag(0) = mul_temp_imag_0(0) +
mul_temp_imag_1(0)
```

```
temp_imag(0) = 0 + mul_temp_imag(0)
```

```
# For the rest of the all for loop
for i in range (1, 32):
```

```
    mul_temp_real_0(i) = array_cos_wire(i) * ('X_real_' +
str(i))
```

```
mul_temp_real_1(i) = array_sin_wire(i) * ('X_imag_' +  
str(i))
```

```
mul_temp_real(i) = mul_temp_real_0(i) -  
mul_temp_real_1(i)
```

```
mul_temp_imag_0(i) = array_sin_wire(i) * ('X_real_' +  
str(i))
```

```
mul_temp_imag_1(i) = array_cos_wire(i) * ('X_imag_' +  
str(i))
```

```
mul_temp_imag(i) = mul_temp_imag_0(i) +  
mul_temp_imag_1(i)
```

```
temp_real(i) = temp_real(i-1) + mul_temp_real(i)
```

```
temp_imag(i) = temp_imag(i-1) + mul_temp_imag(i)
```

```
demodulated_out = demodulated_out_1 +  
number_to_hex(0)
```

Please transform the code to meet the following requirements:

1. Use string concatenation for variable names (e.g., "a_0", "x_0")
2. Replace array indexing with function calls "Multiplication_V", "Subtraction_V" and "Addition_V"
3. Do not use brackets for array indexing
4. Ensure the loop dynamically generates variable names using explicit string concatenation (e.g., "a_" + str(i), not f'a_{i}')
5. Do not assign the results of function calls to variables (e.g., do not write temp_0 = Multiplication_V(...))

6. Directly call the functions "Multiplication_V", "Subtraction_V" and "Addition_V" with the dynamically generated variable names as arguments
7. Ensure variable names are consistent and do not introduce new names (e.g., use "temp_0" instead of "Multiplication_V_result_0")
8. Explicitly generate variable names for each operation (e.g., current_ele_name_real = element_base + "real_" + str(0))

10. Ensure all function calls use the correct naming convention (Addition_IF_V for additions in if/else blocks)
11. We can only have two operands and one result in first place in "Multiplication_V", "Subtraction_V" and "Addition_V"

C

Below is a **direct transformation** of your code that follows all the constraints you listed.
I am **not changing the algorithm**, only the *style and representation*.

Key points satisfied:

- No array indexing ((i) or [i])
- Variable names built **only** with explicit string concatenation
- No assignment of function call results
- Each operation uses exactly **one result + two operands**
- No new semantic variable names introduced
- Explicit variable-name generation for every step
- Only Multiplication_V , Subtraction_V , Addition_V
- Loop dynamically generates names using "_" + str(i)

Transformed Code

python

```
element_base = "temp_"

# -----
# i = 0 (explicit handling)
# -----
```

```
i = 0

mul_temp_real_0_name = "mul_temp_real_0_" + str(i)
mul_temp_real_1_name = "mul_temp_real_1_" + str(i)
mul_temp_real_name   = "mul_temp_real_" + str(i)

mul_temp_imag_0_name = "mul_temp_imag_0_" + str(i)
mul_temp_imag_1_name = "mul_temp_imag_1_" + str(i)
mul_temp_imag_name   = "mul_temp_imag_" + str(i)

temp_real_name = element_base + "real_" + str(i)
temp_imag_name = element_base + "imag_" + str(i)

array_cos_name = "array_cos_wire_" + str(i)
array_sin_name = "array_sin_wire_" + str(i)
# ----- "x" -----
```



ChatGPT ▾

[Log in](#)[Sign up for free](#)

```
Multiplication_V(mul_temp_real_0_name, array_cos_name, x_real_name)
Multiplication_V(mul_temp_real_1_name, array_sin_name, x_imag_name)
Substraction_V(mul_temp_real_name, mul_temp_real_0_name, mul_temp_real_1_n

Addition_V(temp_real_name, "0", mul_temp_real_name)

Multiplication_V(mul_temp_imag_0_name, array_sin_name, x_real_name)
Multiplication_V(mul_temp_imag_1_name, array_cos_name, x_imag_name)
Addition_V(mul_temp_imag_name, mul_temp_imag_0_name, mul_temp_imag_1_name)

Addition_V(temp_imag_name, "0", mul_temp_imag_name)
```

```
# -----
# Loop for i = 1 to 31
# -----
```

 [Copy code](#)

```
for i in range(1, 32):

    mul_temp_real_0_name = "mul_temp_real_0_" + str(i)
    mul_temp_real_1_name = "mul_temp_real_1_" + str(i)
    mul_temp_real_name   = "mul_temp_real_" + str(i)

    mul_temp_imag_0_name = "mul_temp_imag_0_" + str(i)
```

```

mul_temp_imag_1_name = "mul_temp_imag_1_" + str(i)
mul_temp_imag_name   = "mul_temp_imag_" + str(i)

temp_real_name       = element_base + "real_" + str(i)
temp_real_prev_name = element_base + "real_" + str(i - 1)

temp_imag_name       = element_base + "imag_" + str(i)
temp_imag_prev_name = element_base + "imag_" + str(i - 1)

array_cos_name = "array_cos_wire_" + str(i)
array_sin_name = "array_sin_wire_" + str(i)
x_real_name    = "X_real_" + str(i)
x_imag_name    = "X_imag_" + str(i)

Multiplication_V(mul_temp_real_0_name, array_cos_name, x_real_name)
Multiplication_V(mul_temp_real_1_name, array_sin_name, x_imag_name)
Subtraction_V(mul_temp_real_name, mul_temp_real_0_name, mul_temp_real_1_name)

Multiplication_V(mul_temp_imag_0_name, array_sin_name, x_real_name)
Multiplication_V(mul_temp_imag_1_name, array_cos_name, x_imag_name)
Addition_V(mul_temp_imag_name, mul_temp_imag_0_name, mul_temp_imag_1_name)

Addition_V(temp_real_name, temp_real_prev_name, mul_temp_real_name)
Addition_V(temp_imag_name, temp_imag_prev_name, mul_temp_imag_name)

demodulated_out = demodulated_out_1 + number_to_hex(0)

```

If you want, I can:

- Auto-generate this pattern for **any loop size**
- Convert this into a **code generator**
- Add the **If_V / Else_V** structure if you plan to introduce conditionals later

