I have the following code that uses array indexing for multiplication and addition. I want to convert it into a style that uses string concatenation for variable names and replaces array indexing with function calls "Multiplication_V" and "Addition_V". The code should not use brackets for array indexing, and the loop should dynamically generate variable names. Here is the input code:

Input Code:
```
temp = a(0) * array_b_wire(0)
x(0) = temp(0) + 0
for i in range(1, 16):
    temp(i) = a(i) * array_b_wire(i)
    x(i) = temp(i) + x(i-1)
```

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" 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" 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., wire_a = "a_" + str(i)).
9. For conditional logic, use: If_V('c', 'a', 'condition1', ['x'], 'y') for if statements, 'c' represent the comparison value 1, and 'a' represent the comparison value 2. condition can be =,>,<..... 'x' is the name of the variable

that needed in the if/else block (not as output of an equation or the if/else block. e.g. the output of the equation is the first thing that you are in the equation like this 'cc' addtion_V('cc', 'aa', 'bb'). If the content of this ['x'] repeat with 'a' or 'c'. Please not put 'a' or 'c' into it. If without 'a' or 'c' it will be empty, then keep it as [''] 'y' is the output name of the if/else block that is used in the following code. - Value_IF_V(variable, value) for assignments in if
blocks - Addition_IF_V(variable,
operand1, operand2) for additions blocks -
End_IfElse_V("") to end the conditional block 10. Ensure all f
unction 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" 12. no"" when we use "number_to_hex()"

Below is a **direct transformation** of your input code that follows **all stated constraints**. I'm keeping it explicit and mechanical so it's easy to verify and extend.

- ✅ No array indexing (`()` or `[]`)
- ✅ Variable names built **only** with explicit string concatenation
- ✅ No f-strings
- ✅ No assignment of function return values
- ✅ `Multiplication_V` / `Addition_V` used correctly
- ✅ Exactly **two operands + one result**
- ✅ Variable names are explicitly generated and reused
- ✅ No new or implicit variable names introduced
- ✅ No quotes used in `number_to_hex()`
- ✅ Loop dynamically generates names

✅ Transformed Code

python Copy code

```
# ---- i = 0 ----
wire_a = "a_" + str(0)
wire_b = "array_b_wire_" + str(0)
wire_temp = "temp_" + str(0)
wire_x = "x_" + str(0)

Multiplication_V(wire_temp, wire_a, wire_b)
Addition_V(wire_x, wire_temp, number_to_hex(0))


# ---- i = 1 to 15 ----
for i in range(1, 16):

    wire_a = "a_" + str(i)
    wire_b = "array_b_wire_" + str(i)
    wire_temp = "temp_" + str(i)

    wire_x = "x_" + str(i)
    wire_x_prev = "x_" + str(i - 1)

    Multiplication_V(wire_temp, wire_a, wire_b)
    Addition_V(wire_x, wire_temp, wire_x_prev)
```

🔍 Mapping to Original Code

Original	Transformed
`a(i)`	`"a_" + str(i)`
`array_b_wire(i)`	`"array_b_wire_" + str(i)`
`temp(i)`	`"temp_" + str(i)`
`x(i)`	`"x_" + str(i)`
`*`	`Multiplication_V(result, op1, op2)`
`+`	`Addition_V(result, op1, op2)`

If you want:

- an **unrolled version** (no loop at all)
- insertion of **If_V / IfElse logic**
- or conversion to a **code generator script**

just tell me 👍