

## **Artificial Intelligence Solutions for System Design**

### **Dr. Hugh Jack P. E., Western Carolina University**

Dr. Jack is the Cass Ballenger Distinguished Professor of Engineering in the School of Engineering + Technology within Western Carolina University. His interests include robotics, automation, and product design.

### **Dr. Scott Rowe, Western Carolina University**

Scott Rowe is an Assistant Professor in Western Carolina University's School of Engineering + Technology. He joined Western Carolina University in 2021 after studies in concentrated solar power and controls engineering at the University of Colorado Boulder. Scott's research relates to accessible and inexpensive engineering equipment for laboratory education.

# Artificial Intelligence Solutions for Digital Design

## Abstract

Accessible artificial intelligence platforms, especially ChatGPT, are now available to solve engineering questions. Here we evaluate this tool for finite state machine construction in Python. With well-guided queries, ChatGPT built sensible code that implements a microwave oven controller for hardware integration. However, to leverage ChatGPT user knowledge of the programming task was necessary, which included schematics, input, and output delineation, and debug expertise.

Special Note: ChatGPT was not used to author the paper except for items that are shown in the Figures and Program Listing.

## Introduction

In 2022 artificial intelligence became widely accessible through the release of ChatGPT, an interactive platform that produces cogent text in response to user queries.<sup>1</sup> Subsequently, the growth of ChatGPT's popularity, as measured by its user base, quickly outstripped the premier of any prior application.<sup>2</sup> Likely this tool, based on large language models and user feedback, will affect the future of engineering education. By predicting "tokens" (characters or small assemblies of characters) related to a user input the software can:

- Provide simple explanations of technical topics;
- Problem solve;
- Write and debug programs;
- Write and edit reports, memos, and correspondence, and;
- Suggest procedures and methods.

Notably, as a text generation platform, ChatGPT can write computer code in response to student prompts, an ability that could hurt or help nascent engineers grow as programming students.<sup>3</sup> To assess this potential ChatGPT was recently probed with 40 software questions from a programming textbook, where select queries required the authorship or modification of computer code.<sup>4</sup> ChatGPT responses were 44% correct or partially correct.<sup>4</sup> Thus, although ChatGPT can succeed, students should approach artificial intelligence results with discretion. In its current state, the application likely cannot enable the 33% of engineering students known to plagiarize answers, although refinements in ChatGPT might improve the platform's accuracy.<sup>5</sup> Indeed, the large language model AlphaCode, which is currently unavailable, ranks within the top 54.3% of human programmers on code competition challenges.<sup>6</sup> Thus, we expect that eventual improvements will

make artificial intelligence a capable programming tool. Herein we used ChatGPT to develop scalable and structured code. Specifically, ChatGPT was used to construct a finite state machine in the Python computer language for implementation on actual hardware. Putatively, the resultant state machine could drive a microwave oven. This is a higher-order task that requires design planning (state selection) and logical structure.<sup>7</sup> We share lessons learned from ChatGPT use as a development tool in this role. Overall, significant human knowledge of the programming problem remains necessary despite the inclusion of artificial intelligence in this engineering task.

In 2022 AI reached a major milestone with the public release of Dall-E [1] which allowed the general public to create images using plain text requests. This was followed by another product called ChatGPT from the same company.

There has been a long history of artificial intelligence developments, such as the program Eliza, developed in the 1960s [2].

These tools have captured public attention because of their approachability for simple inputs that lead to complex outputs. In the engineering world, these can perform a wide variety of functions including:

- Providing simple explanations of technical topics;
- Problem-solving;
- Writing and debugging programs;
- Writing and editing reports, memos, and correspondence;
- Suggesting procedures and methods;
- Solving simple problems;
- etc.

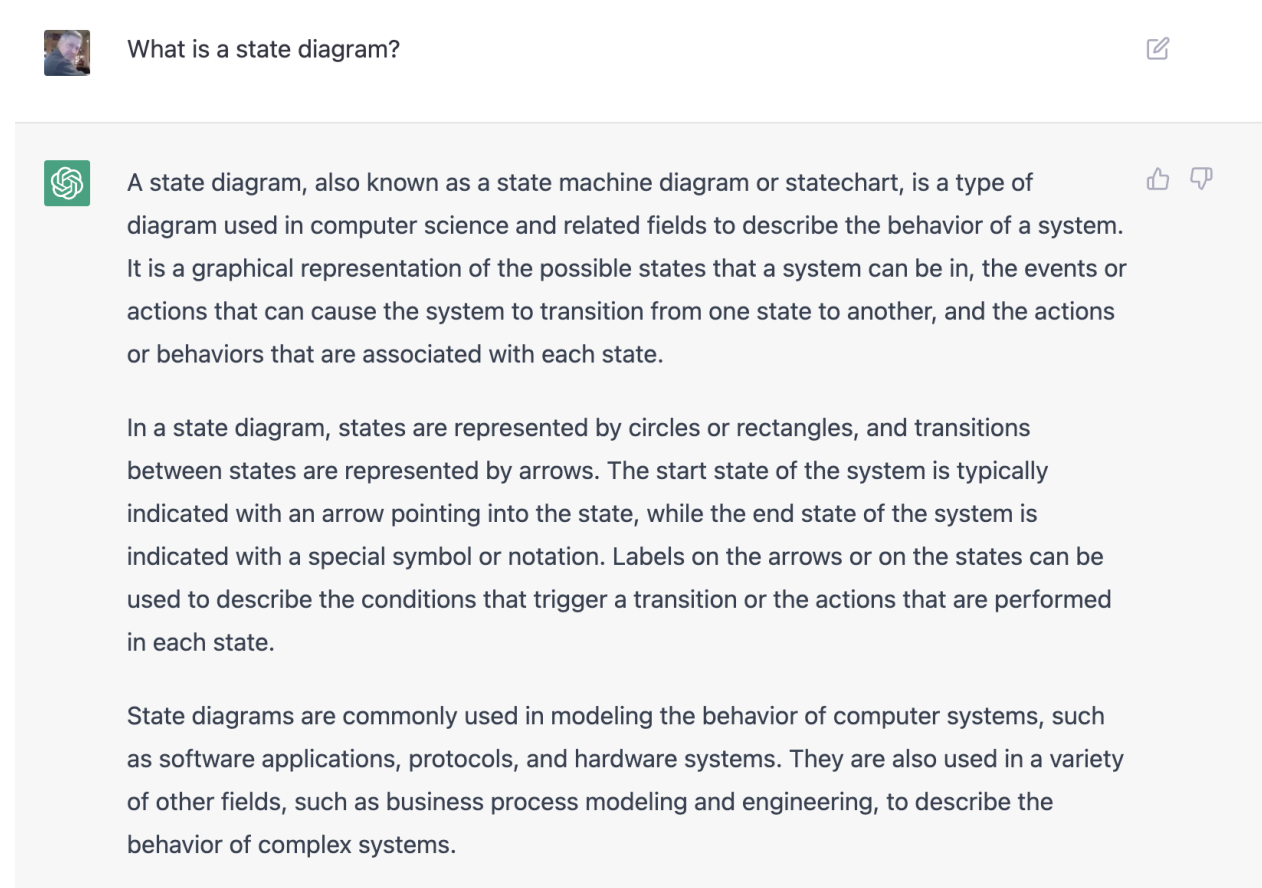
Surprisingly the tools were not developed for solving technical problems and have used less focused training data. We can expect that more focused tools will be developed and available within a timeframe of a few months to years.

Strategically these AI services will become an important part of an engineer's toolbox. However, they will change the way we approach our daily work and the education required to get there.

There are obvious concerns about academic integrity with this tool. In the paper, various examples will be provided to illustrate what the tool can be used for. This should inform the discussion about how it will be used by professionals and how we should prepare our students for the new tool. There will also be a discussion of how our pedagogical methods should be adapted. For illustration, this paper will primarily focus on state diagrams for the examples, but the authors have explored other problem domains with similar results.

## The Basics

ChatGPT searches and trains using common public data sets. It will give very good answers to general questions. In the technical realm, that means many undergraduate problems with written answers are easily found. In Figure 1 a question about state diagrams was quite informative. Currently, the answers are purely text, i.e., no images, figures, graphs, equations, etc. This makes it ideal for just-in-time learning, but it undercuts a traditional form of homework questions. However, students using the same AI engine, e.g. ChatGPT, will have the same or very similar answers.



The screenshot shows a user query: "What is a state diagram?" followed by a detailed AI response. The response is structured with a main definition, a paragraph explaining state representation and transitions, and a paragraph discussing common applications in computer science and other fields.

What is a state diagram?

A state diagram, also known as a state machine diagram or statechart, is a type of diagram used in computer science and related fields to describe the behavior of a system. It is a graphical representation of the possible states that a system can be in, the events or actions that can cause the system to transition from one state to another, and the actions or behaviors that are associated with each state.

In a state diagram, states are represented by circles or rectangles, and transitions between states are represented by arrows. The start state of the system is typically indicated with an arrow pointing into the state, while the end state of the system is indicated with a special symbol or notation. Labels on the arrows or on the states can be used to describe the conditions that trigger a transition or the actions that are performed in each state.

State diagrams are commonly used in modeling the behavior of computer systems, such as software applications, protocols, and hardware systems. They are also used in a variety of other fields, such as business process modeling and engineering, to describe the behavior of complex systems.

**Figure 1 - A Simple Query About State Diagrams**

The tools are always learning but they are not infallible. Figure 2 shows a query made Feb. 3, 2023. The data set in the query is clearly exponential, but ChatGPT guesses that the function is linear and does a simple linear calculation. When the same query was submitted 10 days later, see figure 3, the response was much richer, but it is still trying to fit it to a linear function for a resistor. It does not yet have the ability to identify a squared law device like a diode. This requires that the engineer (or student) be able to critically analyze solutions and troubleshoot when necessary.





What is the equation for;  
V(volts) I(mA)

---

- 1, 2
- 2, 4
- 3, 8
- 4, 16
- 5, 32
- 6, 64



From the data given, the relationship between V and I appears to be proportional, meaning that the ratio of V to I is constant.  

We can find the equation of the line by using the slope-intercept form:

$$y = mx + b$$

where  $y = I$ ,  $x = V$ ,  $m$  is the slope, and  $b$  is the y-intercept.

To find the slope, we can use two points on the line and apply the formula:

$$m = (y_2 - y_1) / (x_2 - x_1)$$

For example, using the points (1, 2) and (2, 4):

$$m = (4 - 2) / (2 - 1) = 2$$

So the equation of the line is:

$$I = 2V$$

This equation shows that the current,  $I$ , is directly proportional to the voltage,  $V$ , with a constant of proportionality of 2 (in mA/V).

**Figure 2 - A Request to fit Data on Feb. 3, 2023**

A second aspect of note is the complex form of the query. Typing a multiline problem statement in a separate editor, and then copying and pasting, allows much richer queries. An essential professional skill will be to create detailed and specific queries. In current practice students and engineers spend a shorter time setting up problems, most of the time executing solution and analysis methods, and very little time verifying solutions. Moving forward, the emphasis will change to problem setup, the suggestion of problem-solving techniques, critical review of results, and troubleshooting. It will be possible to solve more problems in the same amount of time.



The data set you provided appears to be a table of pairs of values, where each pair represents a measurement of voltage (V) and current (I) in milliamperes (mA). To find the equation that describes the relationship between voltage and current, you can try to fit a mathematical model to the data.



One possible model is Ohm's law, which states that the current through a conductor between two points is directly proportional to the voltage across the two points, and inversely proportional to the resistance between them. Mathematically, this can be expressed as:

$$V = IR$$

Where V is the voltage in volts, I is the current in amperes, and R is the resistance in ohms. To convert the current from milliamperes to amperes, we divide by 1000:

$$V = (I/1000)R$$

To find the resistance, we can use two of the data points to solve for R. For example, using the points (2,4) and (3,8), we have:

$$V_1 = (I_1/1000)R$$

$$V_2 = (I_2/1000)R$$

Substituting the values from the data points, we get:

$$2 = (4/1000)R$$

$$3 = (8/1000)R$$

Solving for R, we get:

$$R = 500 \text{ ohms}$$

So, the equation that describes the relationship between voltage and current in this data set is:

$$V = (I/1000) * 500$$

Regenerate response

**Figure 3 - The Same Query as Figure 2 on Feb. 13, 2023**

## Complex Problem Definition

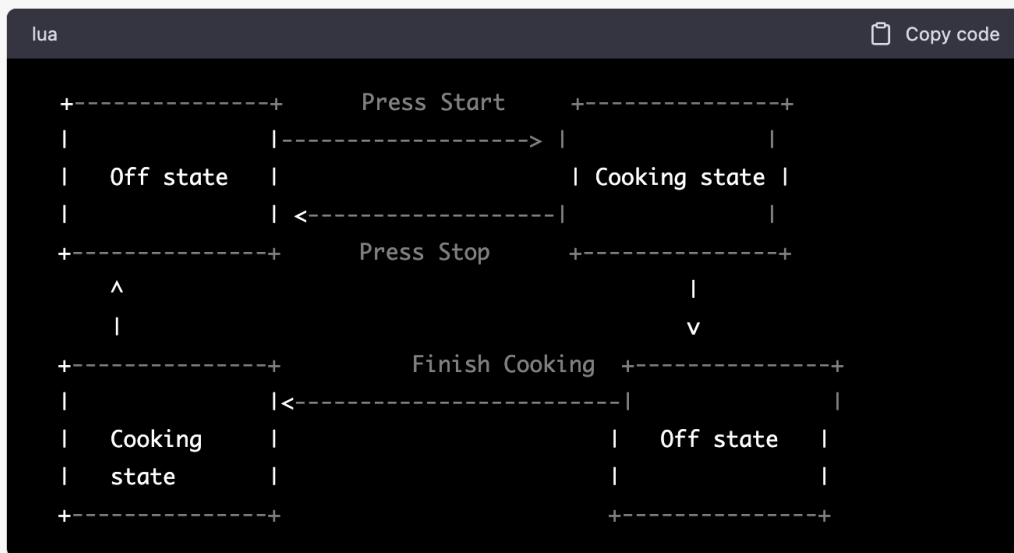
A sample of a state diagram for a microwave query is shown in Figure 4. It is a reasonable solution for a homework problem. But, more work is required to relate it to the inputs, outputs, and a timer for cooking.



what is a reasonable state diagram for a microwave oven?



A state diagram for a microwave oven could look something like this:



This diagram shows the main states that a microwave oven can be in and the events that cause it to transition from one state to another. The states are:

- Off state: The microwave oven is turned off and not in use.
- Cooking state: The microwave oven is cooking food.

The events that trigger transitions between states are:

- Press Start: The user presses the start button to begin cooking.
- Press Stop: The user presses the stop button to stop cooking.
- Finish Cooking: The cooking process is finished.


This state diagram represents a simple, but typical behavior for a microwave oven. In a real-world implementation, there may be additional states and events to represent more complex behavior, such as error handling, different cooking modes, or power levels.

**Figure 4 - Using AI to generate a Problem Definition**


Questions that a student would need to ask, for this solution, are listed below. Figure 5 shows one of the answers to a basic question. The process of asking questions and defining the solution is the major intellectual step of the process.


- Which state is selected during power-on?
- What inputs should I use?
- What outputs should I use?
- How should I set timer values?
- There are transitions without labels, what should I use?
- Why do two states have the same name?
- Etc.

---





which state is selected during power on?





Typically, during power-on of a microwave oven, the initial state selected is the Off state. This means that the oven is turned off and not in use when it is first powered on. The user can then press the start button to begin cooking, which will transition the oven to the Cooking state. From there, the oven will remain in the Cooking state until the cooking process is finished or the user presses the stop button, at which point the oven will transition back to the Off state.

**Figure 5 - Follow-up Questions for the Design**

## Defining the Solution

At some point, enough information is gathered to start thinking about the implementation of the system. If it is a software system this might include a data structure and big-picture flow sequence, like a flowchart.





Provide a data structure called uwave in Python for the state machine:

States: Off, Cooking, Done

Inputs: Start, Stop, Tleft, Door

Outputs: Beeper, Microwaves

Transition: Off to Cooking when Start and not Stop and not Door

Transition: Cooking to Off when Stop or Door

Transition: Cooking to Done when Tleft <= 0

Transition: Done to Off when Stop or Door

First State: Off



Here's an implementation of the state machine in Python using a class-based data structure:



python

Copy code

```
class UWave:
    def __init__(self):
        self.state = "Off"
        self.beeper = False
        self.microwaves = False

    def update(self, start, stop, tleft, door):
        if self.state == "Off":
            if start and not stop and not door:
                self.state = "Cooking"
        elif self.state == "Cooking":
            if stop or door:
                self.state = "Off"
            elif tleft <= 0:
                self.state = "Done"
        elif self.state == "Done":
            if stop or door:
                self.state = "Off"

        self.beeper = self.state == "Done"
        self.microwaves = self.state == "Cooking"
```

Regenerate response

Figure 6 - Python Code for the Core System Design

The AI now has a structure to work with, and it will use again in future answers. The follow-up request is shown in Figure 7 and Listing 1.



Create a python program that reads the inputs, uses uwave to update the states then writes outputs on a raspberry pi zero where,



Input: Start is GPIO 5  
Input: Stop is GPIO 6  
Input: Door is GPIO 13  
Output: Beeper is GPIO 21  
Output Microwaves is GPIO 20



Here is a sample Python program that implements the state machine on a Raspberry Pi Zero using the GPIO (General Purpose Input/Output) pins:



```
python Copy code  
  
import RPi.GPIO as GPIO  
import time  
  
class UWave:  
    def __init__(self):  
        self.state = "Off"  
        self.beeper = False  
        self.microwaves = False
```

**Figure 7 - A Subsequent Request to Add the Hardware Platform Information**

```
import RPi.GPIO as GPIO  
import time  
  
class UWave:  
    def __init__(self):  
        self.state = "Off"  
        self.beeper = False  
        self.microwaves = False  
  
    def update(self, start, stop, tleft, door):  
        if self.state == "Off":  
            if start and not stop and not door:  
                self.state = "Cooking"  
        elif self.state == "Cooking":  
            if stop or door:  
                self.state = "Off"
```

```

        elif tleft <= 0:
            self.state = "Done"
    elif self.state == "Done":
        if stop or door:
            self.state = "Off"

    self.beeper = self.state == "Done"
    self.microwaves = self.state == "Cooking"

GPIO.setmode(GPIO.BCM)
GPIO.setup(5, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(6, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(13, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(21, GPIO.OUT)
GPIO.setup(20, GPIO.OUT)

uwave = UWave()

while True:
    start = not GPIO.input(5)
    stop = not GPIO.input(6)
    door = not GPIO.input(13)
    tleft = 10 # Assume a constant value for tleft for
simplicity
    uwave.update(start, stop, tleft, door)
    GPIO.output(21, uwave.beeper)
    GPIO.output(20, uwave.microwaves)
    time.sleep(0.1)

```

### **Listing 1 - The Full Listing from Figure 7**

At this point, the student/engineer has an almost complete program for the core functionality. Adding a keypad, display, timer function, etc. would require more work but can be done with careful definition of the system. The final stage is testing and troubleshooting. As before, the student/engineer must apply intellectual effort to frame the right questions to probe the system, interpret results, identify issues, query for solutions, and make corrections. Figure 8 shows a possible debugging query for the program. There are a couple of cognitive steps needed to arrive at this point and get a solution.



I tried the program but the Cooking state never ends. What is the problem?



The issue with the program is that there is no mechanism to decrement the `tleft` variable in the `update` function. As a result, the condition `tleft <= 0` is never met, and the state never transitions from Cooking to Done. To fix this, you can add a mechanism to decrement `tleft` in each iteration of the loop, for example:



```
c Copy code  
  
void update(bool start, bool stop, int *tleft, bool door) {  
    switch (state) {  
        case Off:  
            if (start && !stop && !door) {  
                state = Cooking;  
            }  
            break;  
        case Cooking:  
            (*tleft)--;  
            if (stop || door || (*tleft) <= 0) {  
                state = Off;  
            } else if ((*tleft) <= 0) {  
                state = Done;  
            }  
            break;  
        case Done:  
            if (stop || door) {  
                state = Off;  
            }  
            break;  
    }  
    beeper = (state == Done);  
    microwaves = (state == Cooking);  
}
```

**Figure 8 - Debugging the Program**

For further interest, Appendices A and B provide examples of the state machine logic in other languages.

## Improving With Age – Industrial Controls

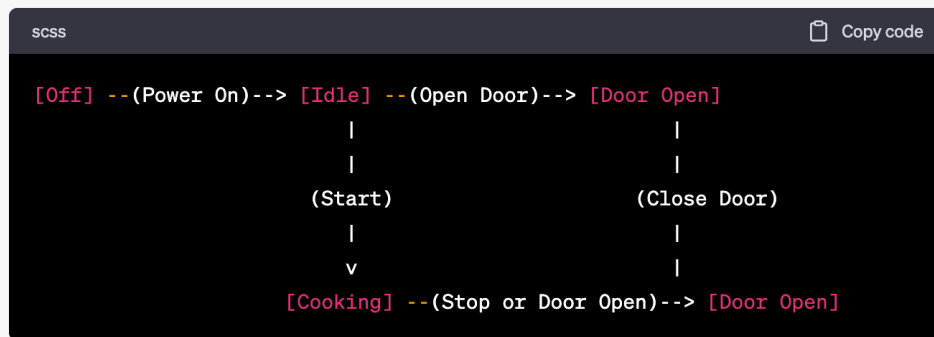
The original examples in this paper were generated with ChatGPT 3.5. This section provides an example that has been generated with ChatGPT 4.0. The query is shown in Figure 9. It is worth noting that the response was much more mature than the response in Figure 4.



create a simple state diagram to control a microwave oven. the inputs should be start and stop. There is a door open sensor



Here's a simplified state diagram for controlling a microwave oven with only Start, Stop, and Door Open inputs:



In this diagram, the microwave oven can be in the following states:

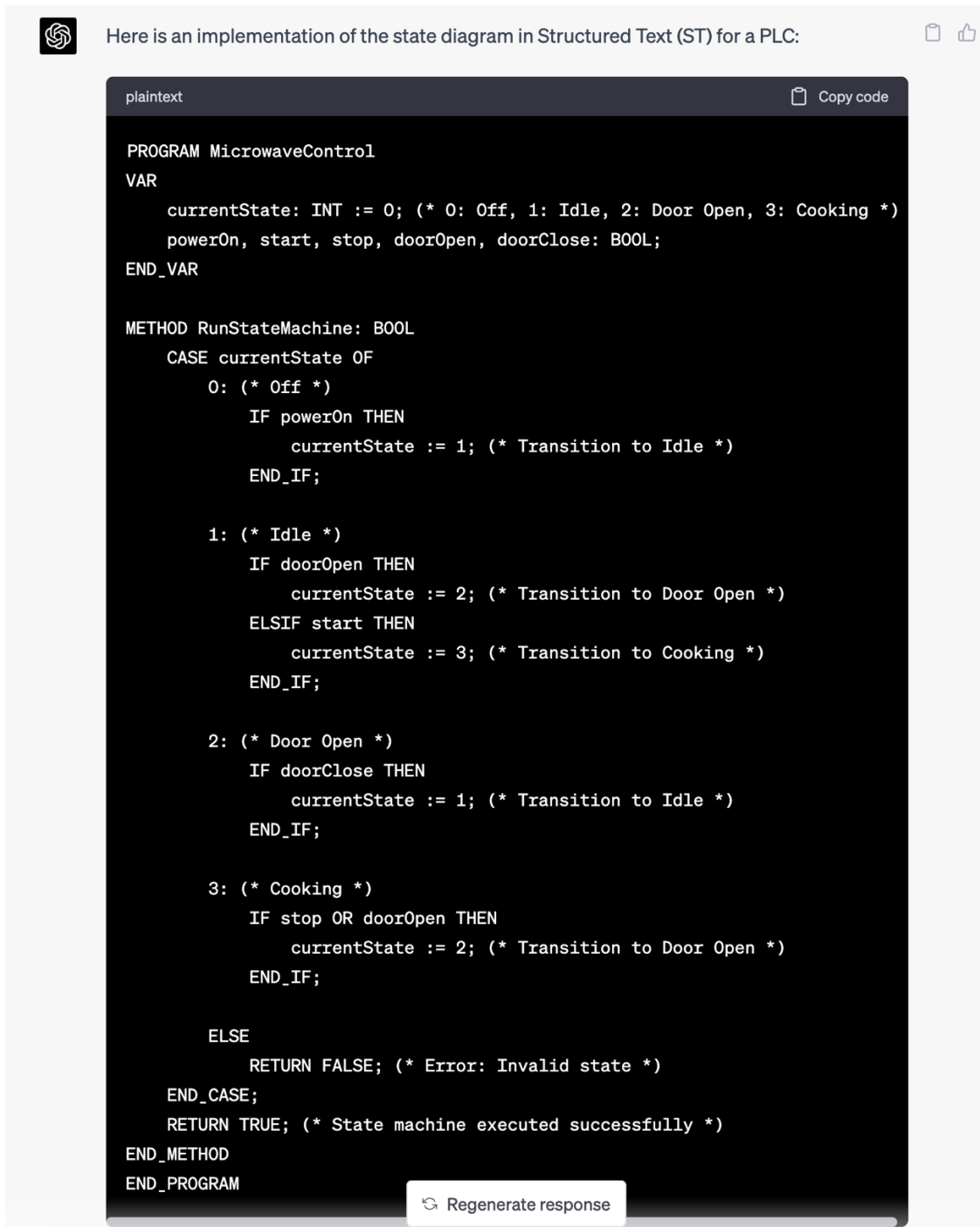
1. Off: The microwave is turned off.
2. Idle: The microwave is on and waiting for user input.
3. Door Open: The microwave door is open.
4. Cooking: The microwave is cooking the food.

Transitions:

- Power On: The microwave is powered on, transitioning from Off to Idle.
- Open Door: The door is opened, transitioning from Idle to Door Open or from Cooking to Door Open (interrupting the cooking process).
- Close Door: The door is closed, transitioning from Door Open to Idle.
- Start: The Start button is pressed, transitioning from Idle to Cooking.
- Stop: The Stop button is pressed or the door is opened, transitioning from Cooking to Door Open.

**Figure 9 – A microwave Oven State Diagram Using ChatGPT 4 on May 1, 2023**

The program language options are also developing. For example, an IEC 61131-2 Structure Text (ST) program is shown in Figure 10. A ladder logic program is shown in Figure 11. These programs do not include all of the elements required for a full industrial control system, but they do provide parts of the program.



```
Here is an implementation of the state diagram in Structured Text (ST) for a PLC:

plaintext Copy code

PROGRAM MicrowaveControl
VAR
    currentState: INT := 0; (* 0: Off, 1: Idle, 2: Door Open, 3: Cooking *)
    powerOn, start, stop, doorOpen, doorClose: BOOL;
END_VAR

METHOD RunStateMachine: BOOL
CASE currentState OF
    0: (* Off *)
        IF powerOn THEN
            currentState := 1; (* Transition to Idle *)
        END_IF;

    1: (* Idle *)
        IF doorOpen THEN
            currentState := 2; (* Transition to Door Open *)
        ELSIF start THEN
            currentState := 3; (* Transition to Cooking *)
        END_IF;

    2: (* Door Open *)
        IF doorClose THEN
            currentState := 1; (* Transition to Idle *)
        END_IF;

    3: (* Cooking *)
        IF stop OR doorOpen THEN
            currentState := 2; (* Transition to Door Open *)
        END_IF;

ELSE
    RETURN FALSE; (* Error: Invalid state *)
END_CASE;
RETURN TRUE; (* State machine executed successfully *)
END_METHOD
END_PROGRAM

Regenerate response
```

**Figure 10 – Structure Text Program for Figure 9 Microwave State Diagram**



write the program in ladder logic



Here is an implementation of the state diagram in Ladder Logic (LD) for a PLC:



```

plaintext Copy code
|--[ ]----[/]---+----- ( )----|
| PowerOn      | Idle      |
|--[/]-----+----- ( )----|
| DoorOpen     | DoorOpen  |

|--[ ]----[/]---+----- ( )----|
| Idle         | Cooking   |
|--[/]----[ ]---+----- ( )----|
| Start        | Cooking   |

|--[ ]----[/]---+----- ( )----|
| DoorOpen     | Cooking   |
|--[/]----[ ]---+----- ( )----|
| Stop         | Cooking   |

|--[ ]----[/]---+----- ( )----|
| Cooking      | Idle      |
|--[/]----[ ]---+----- ( )----|
| DoorClose   | Idle      |

```

This Ladder Logic implementation represents the state transitions based on the inputs (PowerOn, Start, Stop, DoorOpen, DoorClose). The outputs (Idle, DoorOpen, Cooking) represent the current state of the microwave oven. To simplify the diagram, the Off state has been removed, assuming the system starts in the Idle state when powered on.

**Figure 11 – Ladder Logic Program for Figure 9 Microwave State Diagram**

Finally, ChatGPT was asked for a materials list. The result is shown in Figure 12. This illustrates where the value decreases. An experienced designer would not need this list, but it could be useful as a pre-order check list. For students, this list could act as a list on par with hints from a professor.



create a parts list for an industrial controls cabinet for the microwave oven



Here is a basic parts list for an industrial controls cabinet for the microwave oven. The exact components and quantities may vary depending on your specific requirements, but this list should provide a good starting point:



1. Enclosure:
  - Industrial control cabinet (NEMA 4 or IP65 rated) with lock and mounting accessories.
2. PLC (Programmable Logic Controller):
  - PLC with sufficient I/O modules for inputs (Start, Stop, Door Open sensor) and outputs (Door Lock, Magnetron, Fan, and Timer).
  - PLC power supply module.
3. HMI (Human-Machine Interface):
  - Industrial touchscreen HMI panel for operator input and system status monitoring.
  - HMI communication module/cable for PLC connectivity.
4. Sensors and Switches:
  - Door Open/Close sensor (magnetic or proximity switch).
  - Temperature sensor (RTD or thermocouple) for monitoring the oven cavity temperature.
  - Optional: Current transformer (CT) to monitor the magnetron's power consumption.
5. Actuators and Control Devices:
  - Contactor or relay for magnetron control.
  - Contactor or relay for fan control.
  - Contactor or relay for door lock control (safety interlock).
6. Power Distribution and Protection:
  - Circuit breaker or fuse for main power protection.
  - Circuit breakers or fuses for individual components (PLC, HMI, magnetron, fan, door lock).
  - Terminal blocks for power distribution and wiring connections.
  - Power supply for sensors and other low voltage devices.
7. Wiring and Connectivity:
  - Shielded cables for sensor and actuator connections.
  - Control and communication cables for PLC and HMI.

**Figure 12 – Controls Cabinet Parts List (Partial) for Microwave Control**

## Conclusion

The tools decrease the time required to perform tedious tasks like writing reports and algorithms. Instead, the engineer must be able to define a problem clearly for the AI tools. For example, more time should be spent on specifications and concepts and less time on detailing. But, the engineers must develop better test plans to verify the work of the AI.

As educators, we have focused on the ‘how of problem solving’ with methods such as mesh current analysis, method of joints in trusses, and solving differential equations. The new emphasis will need to become the ‘why of problem solving’ with more time setting up schematics, free-body diagrams, and other models. But instead of spending time solving problems, students will need to spend their time testing and verifying models.



## References

1. Thorp, H. H. *ChatGPT is fun, but not an author*. *Science* vol. 379 313–313 (American Association for the Advancement of Science, 2023).
2. Dennean, K., Gantori, S., Lima K., D., Pu, A. & Gilligan, R. *Let's Chat About ChatGPT*. (2023).
3. Rowe, S. C. & Nuttelman, C. R. A MATLAB Assignment Framework for Engineering Education that Automates Grading. (2022).
4. Jalil, S., Rafi, S., LaToza, T. D., Moran, K. & Lam, W. ChatGPT and Software Testing Education: Promises & Perils. *arXiv preprint arXiv:2302.03287* (2023).
5. Bertram Gallant, T., Van Den Einde, L., Ouellette, S. & Lee, S. A systemic analysis of cheating in an undergraduate engineering mechanics course. *Science and engineering ethics* 20, 277–298 (2014).
6. Li, Y. *et al.* Competition-level code generation with alphacode. *Science* 378, 1092–1097 (2022).
7. Rowe, S. C., Samson, C. I. & Clough, D. E. A framework to guide the instruction of industrial programmable logic controllers in undergraduate engineering education. *Education for Chemical Engineers* 31, 76–84 (2020).
8. openai.com, last accessed May 1, 2023.
9. Oettinger, A. G., “Eliza - A Computer Program For the Study of Natural Language Communication Between Man and Machine”, *Communications of the ACM*, January 1966, Vol. 9, Num. 1, pg. 36-45.
10. Brady, J., “AI-designed Chips Reach Scale with First 100 Commercial Tape-outs Using Synopsys Technology”, <https://news.synopsys.com/2023-02-07-AI-designed-Chips-Reach-Scale-with-First-100-Commercial-Tape-outs-Using-Synopsys-Technology?s=31>, accessed, February 22, 2023
11. Williams, T., “Some companies are already replacing workers with ChatGPT, despite warnings it shouldn't be relied on for ‘anything important’”, *Fortune Magazine Online*, <https://fortune.com/2023/02/25/companies-replacing-workers-chatgpt-ai/>, accessed February 28, 2023

## Appendix A - The State Machine Program in VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity uwave is
  port(
    start  : in  std_logic;
    stop   : in  std_logic;
    tleft  : in  integer;
    door   : in  std_logic;
    beeper : out std_logic;
    microwaves : out std_logic
  );
end entity uwave;

architecture behavior of uwave is
  type state_type is (Off, Cooking, Done);
  signal state : state_type := Off;
begin
  process(start, stop, tleft, door)
  begin
    case state is
      when Off =>
        if start = '1' and stop = '0' and door = '0' then
          state <= Cooking;
        end if;
      when Cooking =>
        if stop = '1' or door = '1' or tleft <= 0 then
          state <= Off;
        elsif tleft <= 0 then
          state <= Done;
        end if;
      when Done =>
        if stop = '1' or door = '1' then
          state <= Off;
        end if;
    end case;
  end process;

  beeper <= (state = Done);
  microwaves <= (state = Cooking);
end architecture behavior;
```

## Appendix B - The State Machine Program in C

```
#include <stdbool.h>

typedef enum {
    Off,
    Cooking,
    Done
} State;

State state = Off;
bool beeper = false;
bool microwaves = false;

void update(bool start, bool stop, int tleft, bool door) {
    switch (state) {
        case Off:
            if (start && !stop && !door) {
                state = Cooking;
            }
            break;
        case Cooking:
            if (stop || door || tleft <= 0) {
                state = Off;
            } else if (tleft <= 0) {
                state = Done;
            }
            break;
        case Done:
            if (stop || door) {
                state = Off;
            }
            break;
    }
    beeper = (state == Done);
    microwaves = (state == Cooking);
}
```