Group 13
Blocks ‘O Code
Midterm Demo

John Gierlach, Robert Buch, Tyler Goldsmith, Nandhu Jani

Hardware Implementation

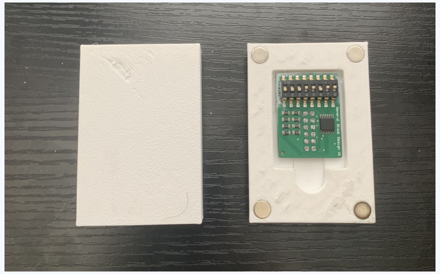
3x3 Grid Video

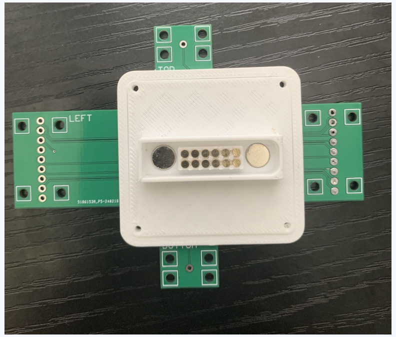
- Build out the 4x4 grid with the blocks
- Describe what each block represents
- Briefly explain how the data is shifting throughout the grid
- Briefly explain how the ESP32 is driving the logic
- Briefly explain how the ESP32 is receiving data

Connector Design

1. Block design
   a. 3D printed magnetic box
   b. Dataflow PCB

2. Grid Design
   a. 3D printed connector/aligner
   b. Grid PCB





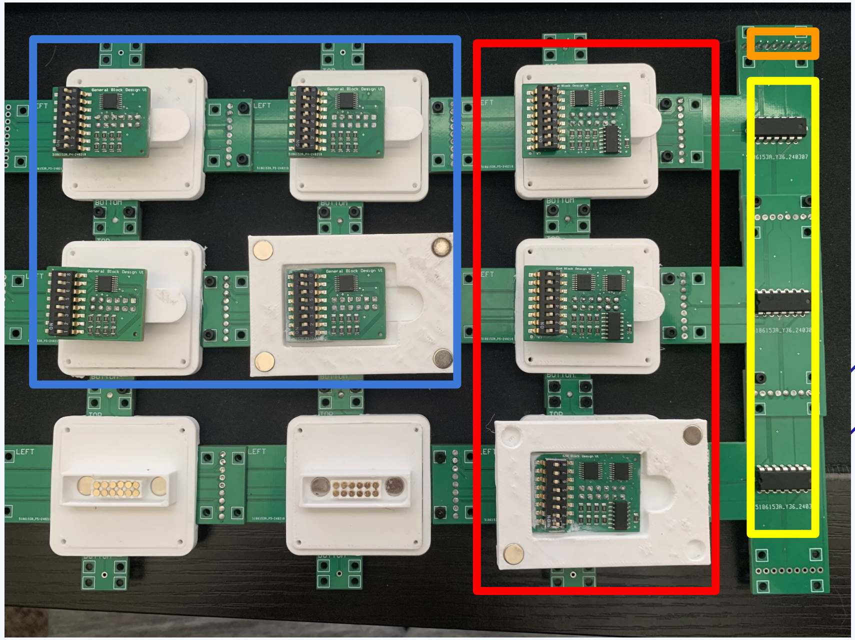
3x3 Grid Design

1. End Blocks | {} ;
   a. Shifts address to ESP

2. Main Blocks
   a. Passes address to end block

3. Power Control Logic Gates

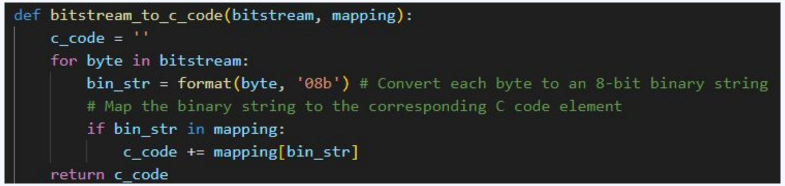
4. Data Pins to ESP32



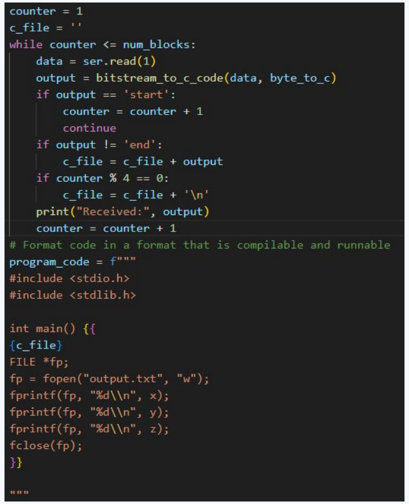
Software Implementation

C File Generation Demo

Add video scrolling through code to explain code generation process, give example of running that part of the script independently



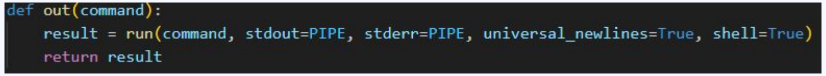
```python
def bitstream_to_c_code(bitstream, mapping):
    c_code = ''
    for byte in bitstream:
        bin_str = format(byte, '08b') # Convert each byte to an 8-bit binary string
        # Map the binary string to the corresponding C code element
        if bin_str in mapping:
            c_code += mapping[bin_str]
    return c_code
```



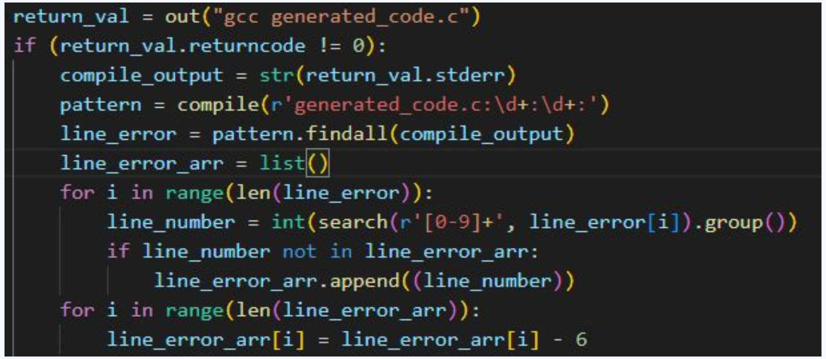
```python
counter = 1
c_file = ''
while counter <= num_blocks:
    data = ser.read(1)
    output = bitstream_to_c_code(data, byte_to_c)
    if output == 'start':
        counter = counter + 1
        continue
    if output != 'end':
        c_file = c_file + output
    if counter % 4 == 0:
        c_file = c_file + '\n'
    print("Received:", output)
    counter = counter + 1
# Format code in a format that is compilable and runnable
program_code = f"""
#include <stdio.h>
#include <stdlib.h>

int main() {{
{c_file}
FILE *fp;
fp = fopen("output.txt", "w");
fprintf(fp, "%d\\n", x);
fprintf(fp, "%d\\n", y);
fprintf(fp, "%d\\n", z);
fclose(fp);
}}

"""
```

C File Compilation Demo

Add video scrolling through compilation and output capture, show running that part of the script on two files (one working, one not) and spit that output to the command line



```python
def out(command):
    result = run(command, stdout=PIPE, stderr=PIPE, universal_newlines=True, shell=True)
    return result
```



```python
return_val = out("gcc generated_code.c")
if (return_val.returncode != 0):
    compile_output = str(return_val.stderr)
    pattern = compile(r'generated_code.c:\d+:\d+:')
    line_error = pattern.findall(compile_output)
    line_error_arr = list()
    for i in range(len(line_error)):
        line_number = int(search(r'[0-9]+', line_error[i]).group())
        if line_number not in line_error_arr:
            line_error_arr.append((line_number))
    for i in range(len(line_error_arr)):
        line_error_arr[i] = line_error_arr[i] - 6
```

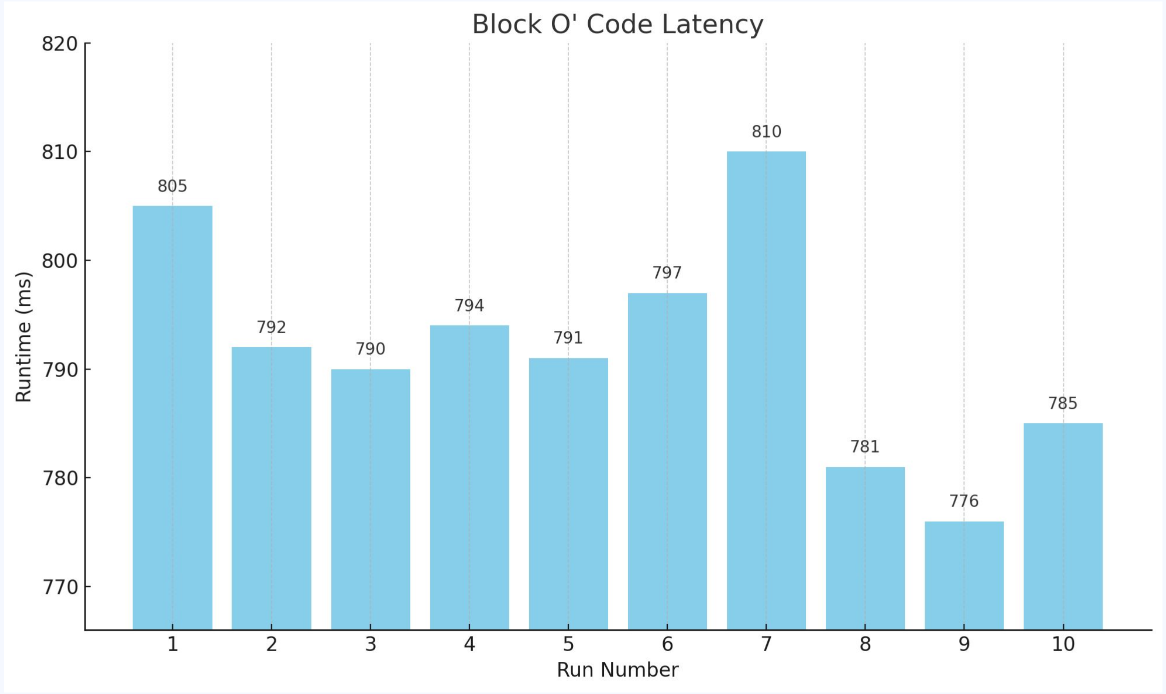
Full System Test Video

Engineering Specifications

Specifications & Requirements

Requirements	***Descriptions***
Feedback Latency	Less than or equal to 5 seconds
Block Durability	1 Meter drop
Block Types	15 Addressable 8-bit Codes
Block Limits	Minimum 16 Blocks
Power	5V @ 2A
Inactivity	Shutdown after 30 Seconds of non-usage

Spec #1: Feedback Latency



Mean: 792.1 ms
Variance: ~95.29 ms²

Spec #3: Block Durability

Add video of dropping block from table height and then place it on the board and show that data still transmits to the pi

Spec #2: Block Types

Add video showing pi receiving 16 different addresses, show terminal output from pi with translated data

Next Steps

- Build the cosmetic portion of the project
  - Wooden board (laser cut)
  - Improved block chassi
  - External LEDs
- Receive and construct embedded PCB with ESP-32 and Raspberry Pi
- Expand grid to 4x4 to allow for more interesting code
- Minor bug/reliability fixes
  - Connector redesign, some connectors are loose
  - Minor code improvements

Thank you!

Question?