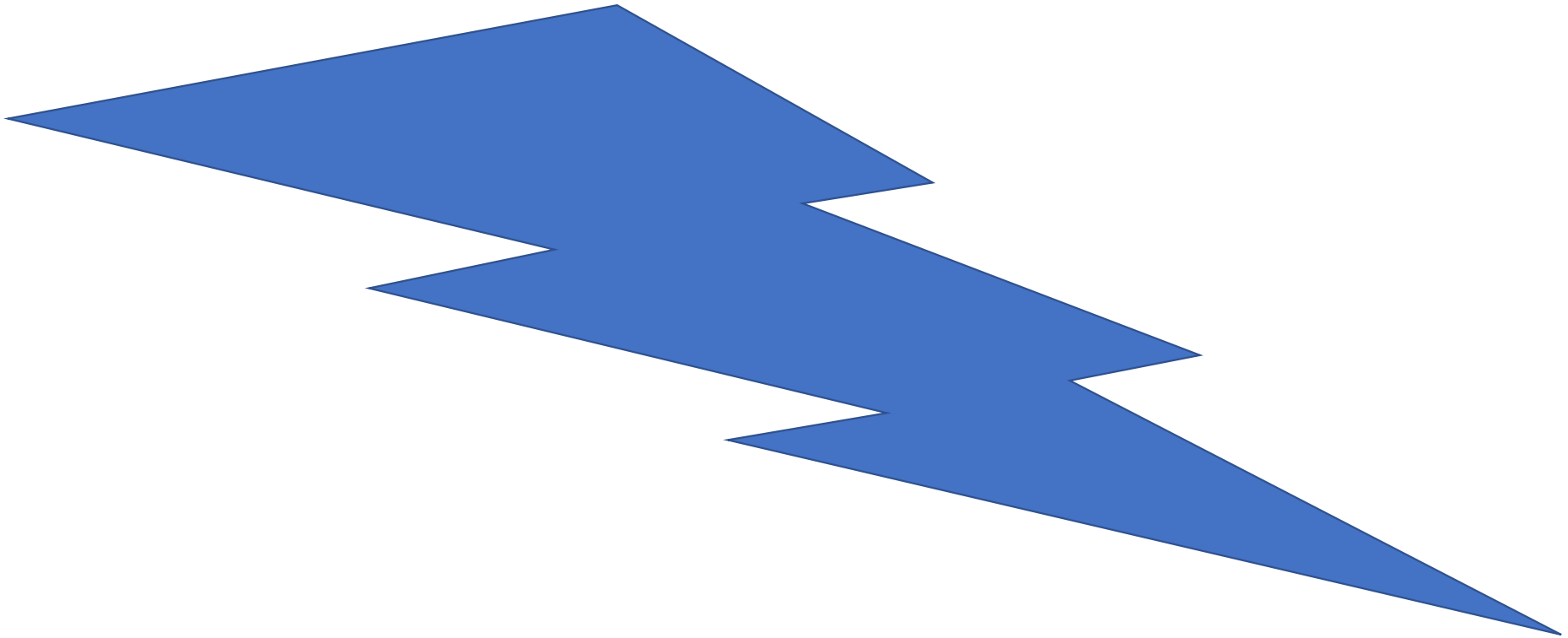
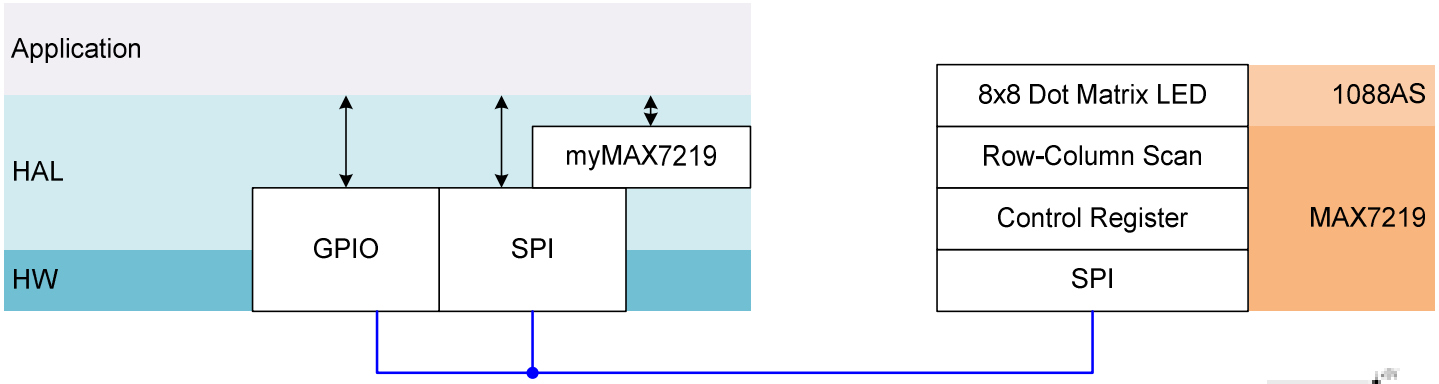
Lab. 2
- SPI
- connect SPI to MAX7219
- array variable and constant
- multiple source files
- 2-dimension array
- variables for inter-process interfacing
- multiple task rates

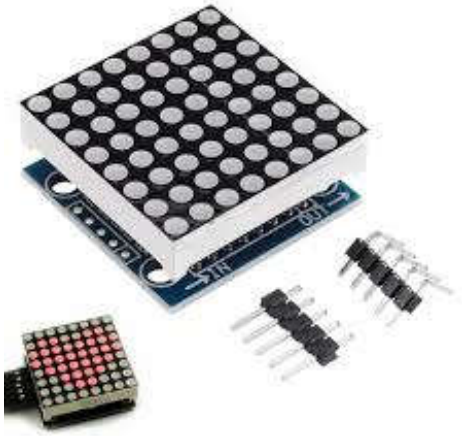
L02p01

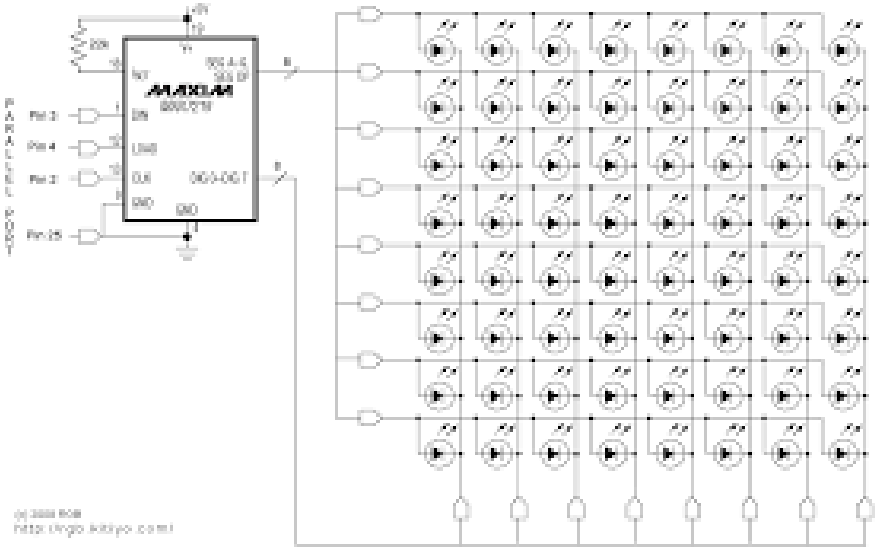


Background knowledge

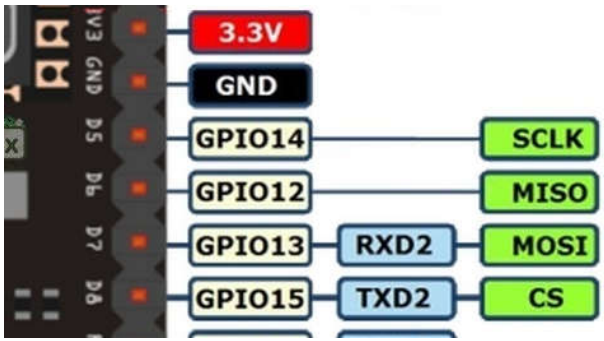


8x8 Dot Matrix LED	1088AS
Row-Column Scan	MAX7219
Control Register	
SPI	



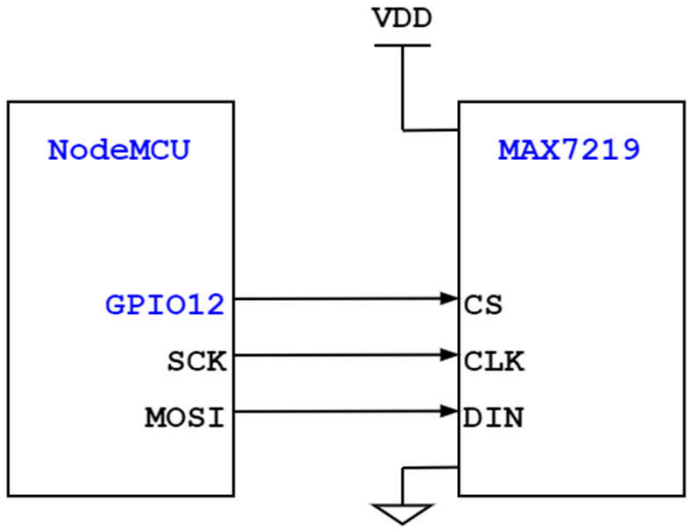


Connect MAX7219 to ESP8266



L02p01.ino > myschedule()

```
1   #include <Arduino.h>
2   #include <SPI.h>
3   #include "myMAX7219.h"
4
5   const int CS_PIN   = 12;
6   const int DIN_PIN  = 13;
7   const int CLK_PIN  = 14;
```



Initialize, myMAX7219 API

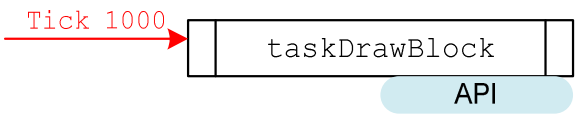
```
17  void setup() {
18      delay(3000);
19      // H/W initialize
20      SPI.begin();
21      SPI.setBitOrder(MSBFIRST);
22      SPI.setClockDivider(SPI_CLOCK_DIV16);
23      SPI.setDataMode(SPI_MODE0);
24      pinMode(CS_PIN, OUTPUT);
25      digitalWrite(CS_PIN, HIGH);
26      MAX7219_init();
27      // Variables and state initialize
28      tick_last = millis();
29      // End of setup
30      MAX7219_write_reg(REG_SHUTDOWN, 0x01);
31  }
```

myMAX7219.h > ...

```
1
2   #define REG_DIGIT(x)      (0x1+(x))
3   #define REG_DECODE_MODE   (0x9)
4   #define REG_INTENSITY     (0xA)
5   #define REG_SCAN_LIMIT    (0xB)
6   #define REG_SHUTDOWN      (0xC)
7   #define REG_DISPLAY_TEST  (0xF)
8
9   extern const int CS_PIN;
10  void MAX7219_write_reg(uint8_t addr, uint8_t data);
11  void MAX7219_init(void);
```

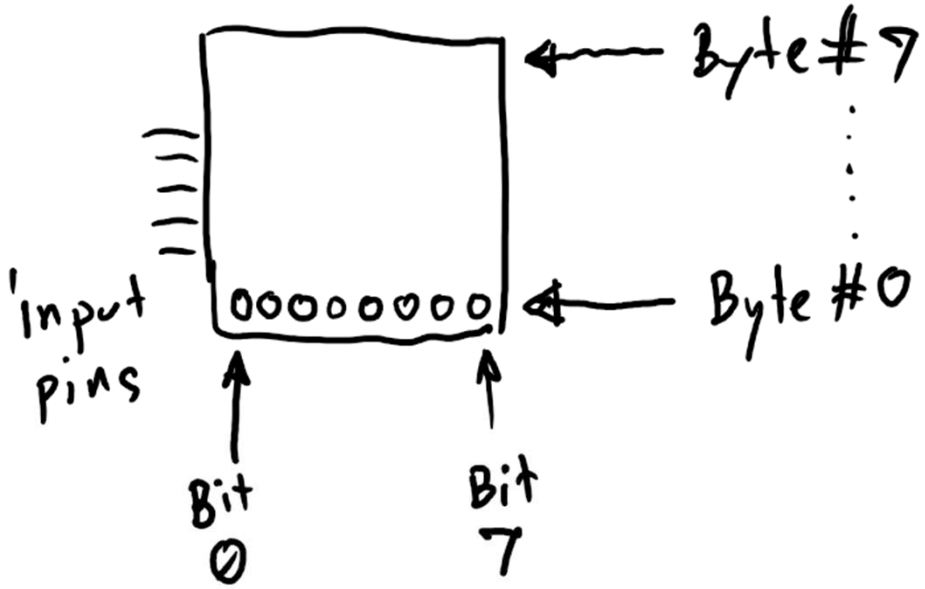
Application: taskDrawBlock

```
32  void loop() {
33      myschedule();
34      if (rqtask == 1) {
35          rqtask = 0;
36          taskDrawBlock();
37      }
38  }
```



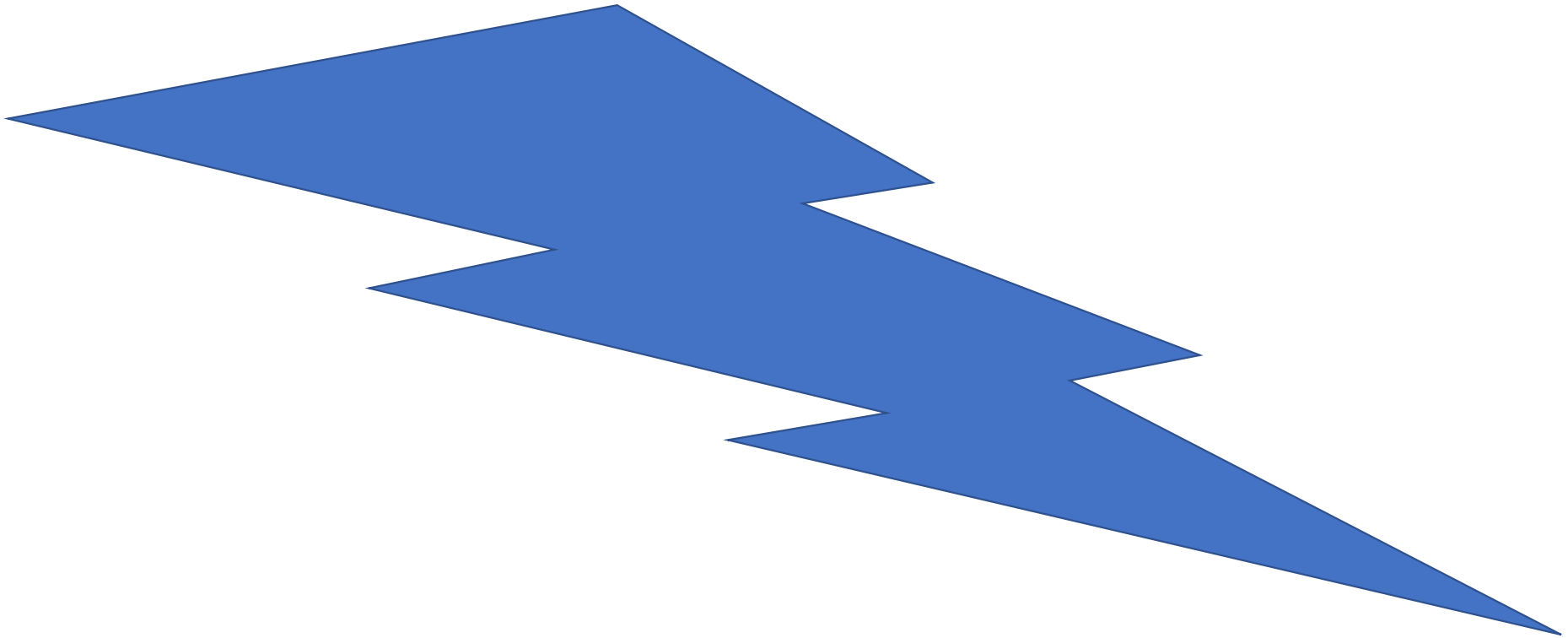

```
58  const byte pattern[8] = {
59      0B00111000,
60      0B00111000,
61      0B00111000,
62      0B00111000,
63      0B11111110,
64      0B01111100,
65      0B00111000,
66      0B00010000
67  };
68  void taskDrawBlock() {
69      int i;
70      for (i = 0; i < 8; i++) {
71          MAX7219_write_reg(REG_DIGIT(i), pattern[i]);
72      }
73  }
```

taskDrawBlock




```
int i;
for (i = 0; i < 8; i++) {
    MAX7219_write_reg(REG_DIGIT(i), pattern[i]);
}
```

L02p02



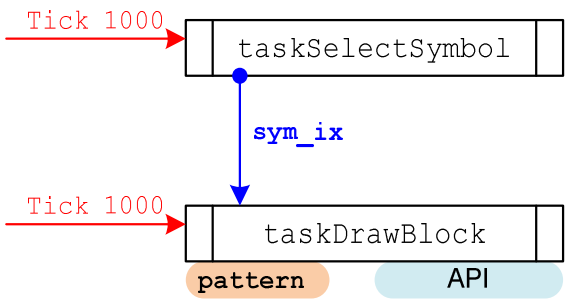
L02p02: Font ROM

```
2   const byte pattern[][8] = {
3   //Up
4   {   0B00011100,
5       0B00011100,
6       0B00011100,
7       0B00011100,
8       0B01111111,
9       0B00111110,
10      0B00011100,
11      0B00001000 },
12  //Down
13  {   0B00001000,
14      0B00011100,
15      0B00111110,
16      0B01111111,
17      0B00011100,
18      0B00011100,
19      0B00011100,
20      0B00011100 },
```

```
39  // 3
40  {   0B00111110,
41      0B01111111,
42      0B01100011,
43      0B01110000,
44      0B00110000,
45      0B01100011,
46      0B01111111,
47      0B00111110 },
48  //4
49  {   0B00110000,
50      0B00110000,
51      0B01111111,
52      0B01111111,
53      0B00110011,
54      0B00110110,
55      0B00111100,
56      0B00111000 }
57  };
```

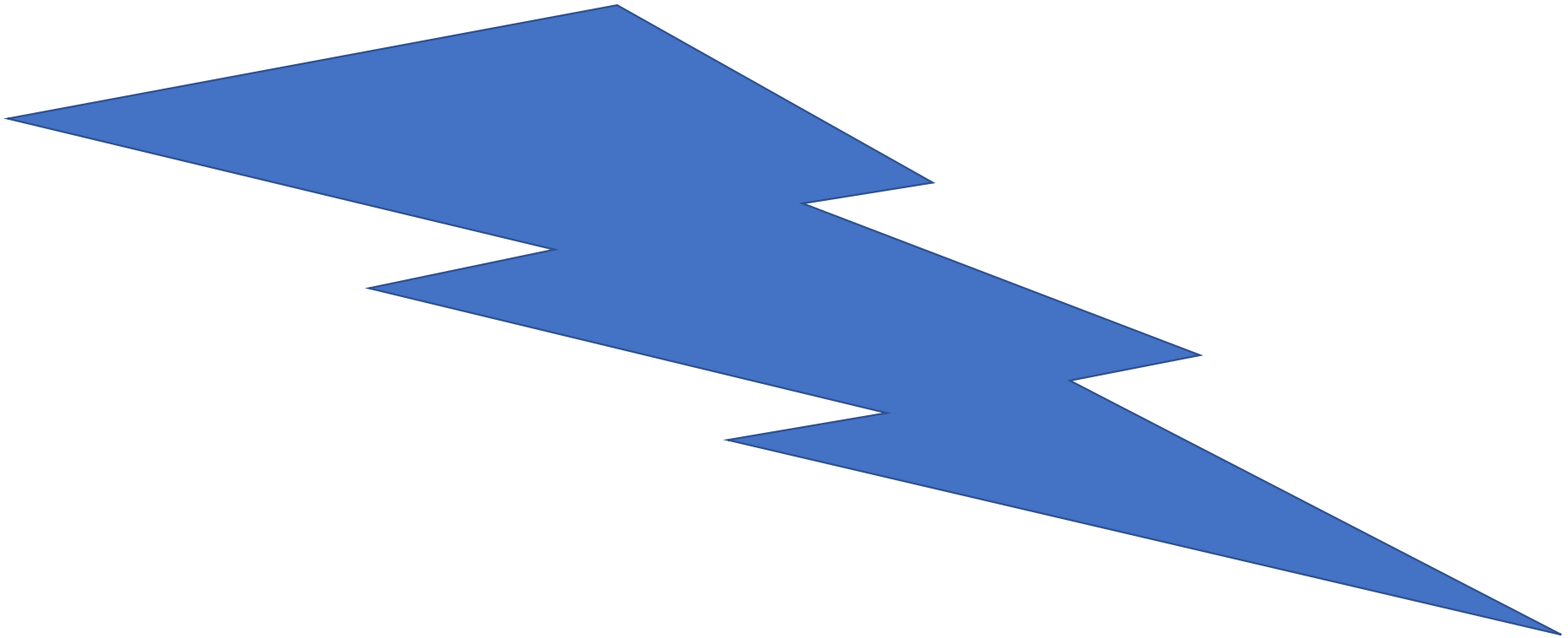
Use pattern form font ROM, Task synchronization

```
38  void loop() {
39      myschedule();
40      if (rqtask == 1) {
41          rqtask = 0;
42          taskSelectSymbol();
43          taskDrawBlock();
44      }
45  }
```

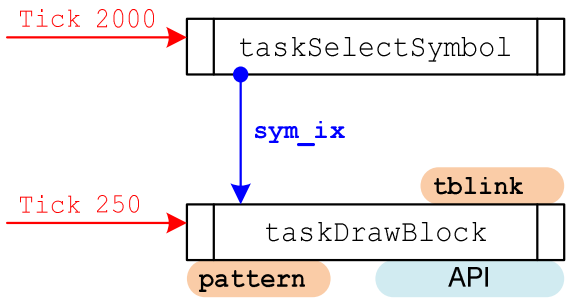



```
65  void taskDrawBlock() {
66    int i;
67    for (i = 0; i < 8; i++) {
68        MAX7219_write_reg(REG_DIGIT(i), pattern[sym_ix][i]);
69    }
70  }
71  void taskSelectSymbol() {
72    sym_ix++;
73    if (sym_ix == 6) sym_ix = 0;
74  }
```

L02p03



L02p03: Blinking effect

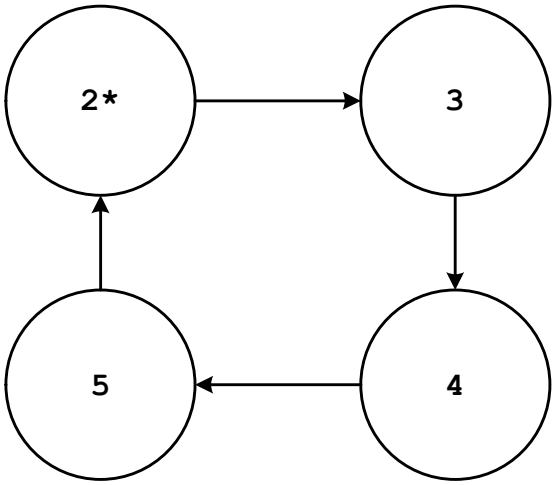



```
40  void loop() {
41      myschedule();
42      if (rqtask == 1) {
43        rqtask = 0;
44        taskDrawBlock();
45      }
46      if (rqCount == 1) {
47        rqCount = 0;
48        taskSelectSymbol();
49      }
50  }
```

```
90  void taskSelectSymbol() {
91    sym_ix++;
92    if (sym_ix == 6) sym_ix = 2;
93  }
```

```
76  void taskDrawBlock() {
77    int i;
78    if (tblink == 1) {
79      for (i = 0; i < 8; i++) {
80          MAX7219_write_reg(REG_DIGIT(i), pattern[sym_ix][i]);
81      }
82    }
83    else {
84      for (i = 0; i < 8; i++) {
85          MAX7219_write_reg(REG_DIGIT(i), 0);
86      }
87    }
88    tblink ^= 1;
89  }
```

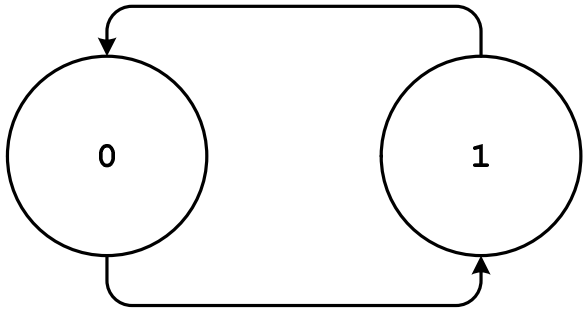
taskSelectSymbol: State



State variable: sym_ix

- “sym_ix” is state variable and index of pattern of ‘1’ to ‘4’.
- State of 2 is initial state.
- State is changed every 2 seconds.

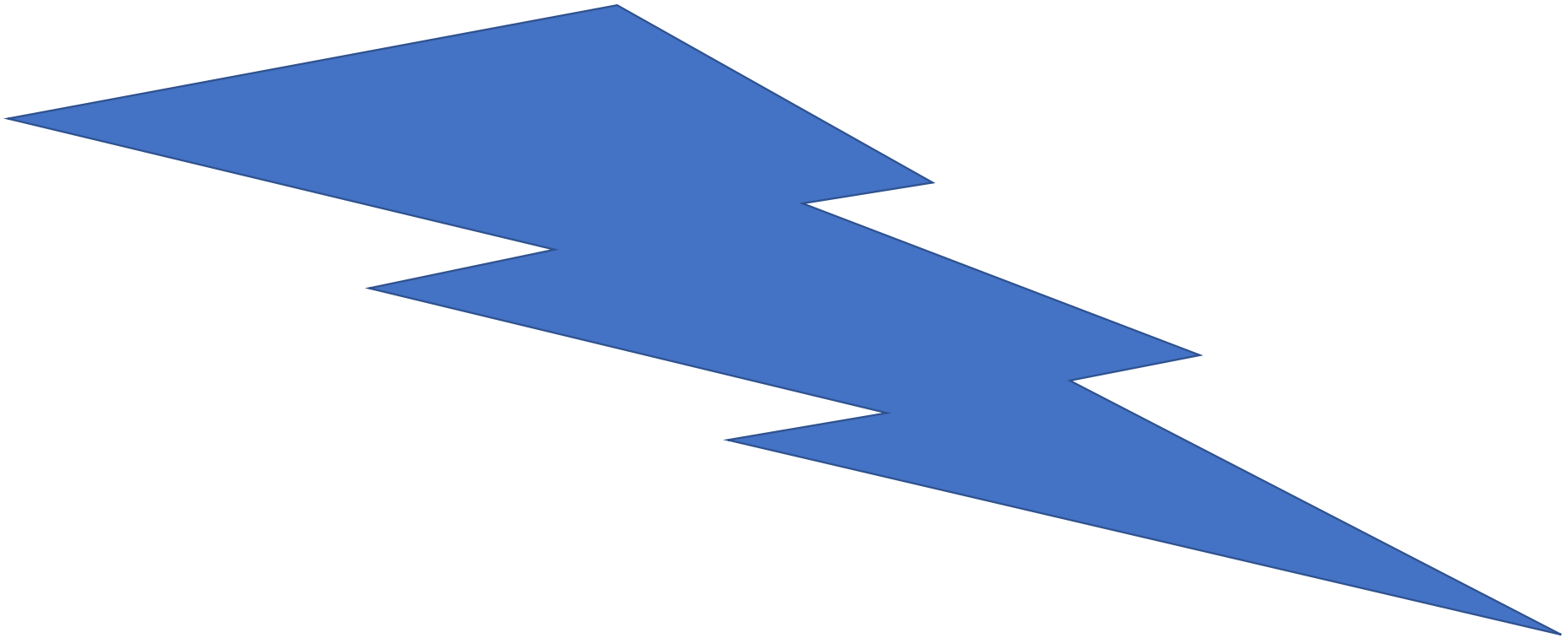
taskDrawBlock: Blinking



`State variable: tblink`

- State 0:
  - Task: off all LEDs
  - Next state: 1, -
- State 1:
  - Task: display pattern[sym_ix]
  - Next state: 0, -

L02p04



L02p04: Another effect, design flow concept

Analyzing and Learning to make understanding
- Hardware (Schematic)
- Source code
- Model (Task synchronization, Task schedule)
- Result

Design and Implement
- Requirement (the result must meet the requirement)
- Hardware (HW must carry on functions delivered from the requirements)
- Model and Task synchronization
  - task definition, resource allocation
  - task schedule
  - task synchronization
  - flow chart for an individual task
  - FSM for a task with state operation
- Coding: write source codes and debug them
  - (keep in mind, you cannot write a program in one time to meet all requirements without luck.)

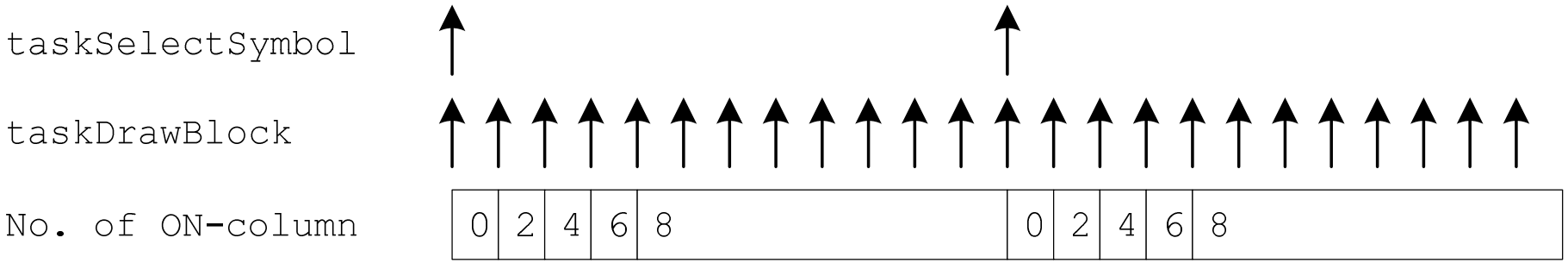
Get requirements and extract functions

Get requirements: see a clip

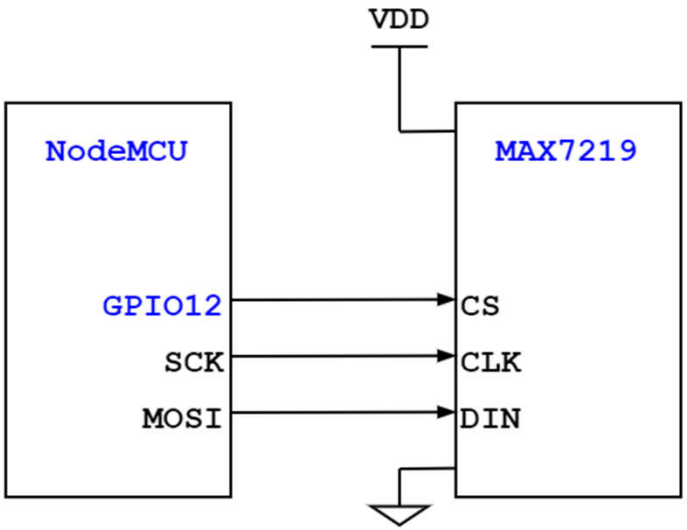
Extracted function
- Four pages to be displayed {1, 2, 3, 4}, and repletely run
- Each page has 3 seconds for displaying
- There are 5 states in one page displaying
  - All columns are off. It is in this state for 250 ms.
  - 2 left-columns are on, the rest are off. It is in this state for 250 ms.
  - 4 left-columns are on, the rest are off. It is in this state for 250 ms.
  - 6 left-columns are on, the rest are off. It is in this state for 250 ms.
  - All columns are on. It is in this state until page display ends.

Please ensure that extracted functions are match to the requirements.

Extracted Function



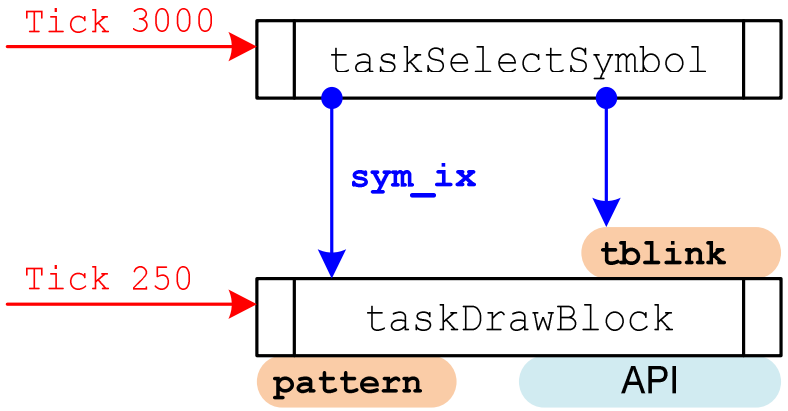
Hardware design



Modeling

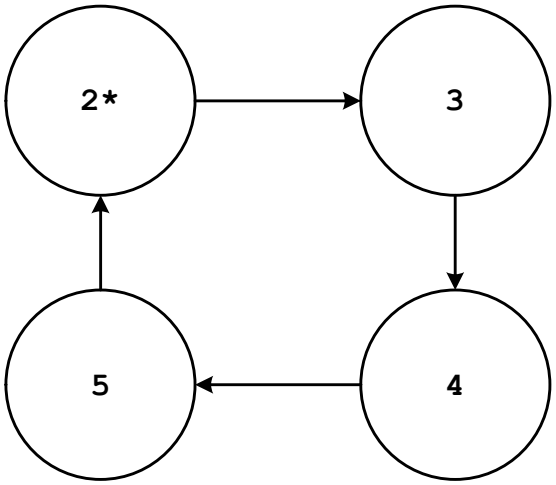
- task definition, resource allocation
- task schedule
- task synchronization
- flow chart for an individual task
- FSM for a task with state operation

Modeling



- “sym_ix” selects pattern to be displayed.
- “tblink” indicates number of columns to be turn on.

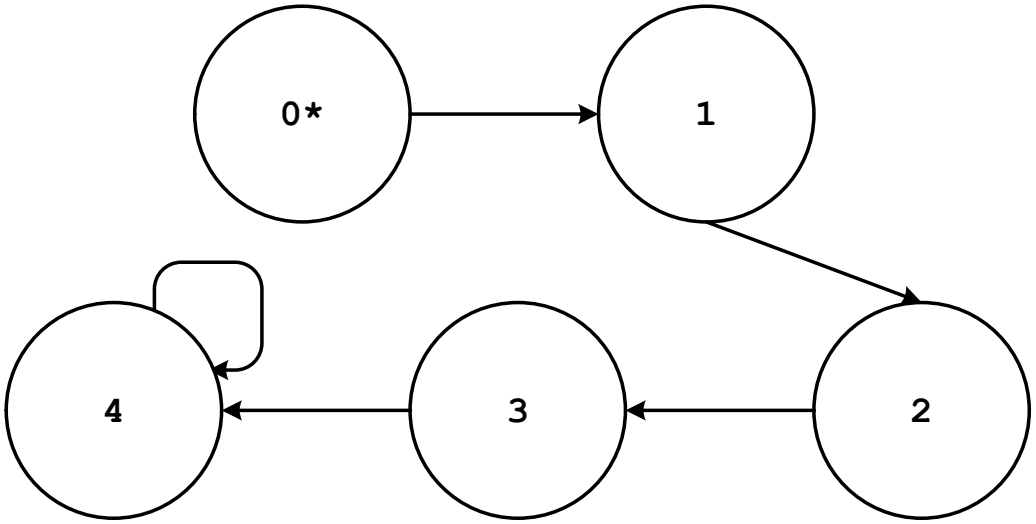
taskSelectSymbol: State




```
State variable: sym_ix
```

- “sym_ix” is state variable and index of pattern of ‘1’ to ‘4’.
- State of 2 is initial state.
- State is changed every 3 seconds.
- When state changes, “tblink” is reset.

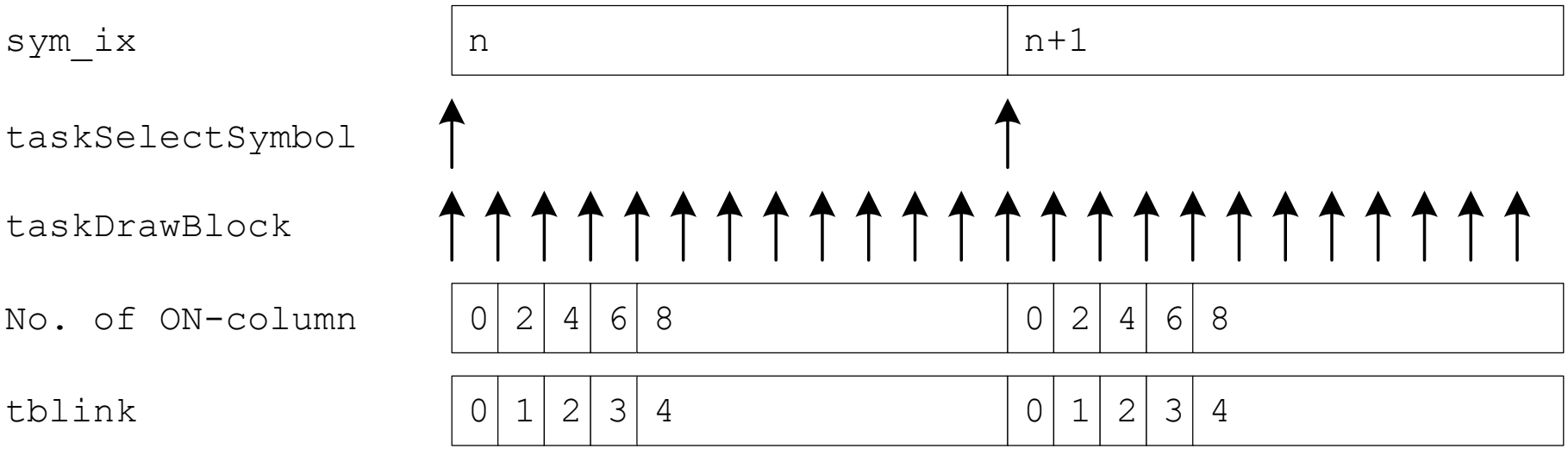
taskDrawBlock: State



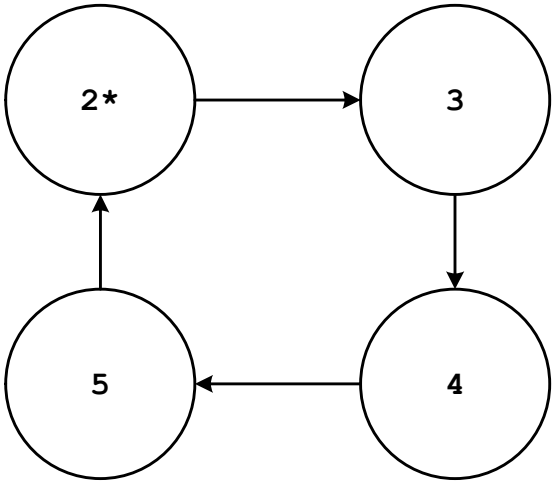

```
State variable: tblink
```

- "tblink" is state variable and is reset by "taskSelectSymbol"
- State
  - 0 – 0 column ON
  - 1 – 2 columns ON
  - 2 – 4 columns ON
  - 3 – 6 columns ON
  - 4 – 8 columns ON

Recheck Model vs. Extracted Function



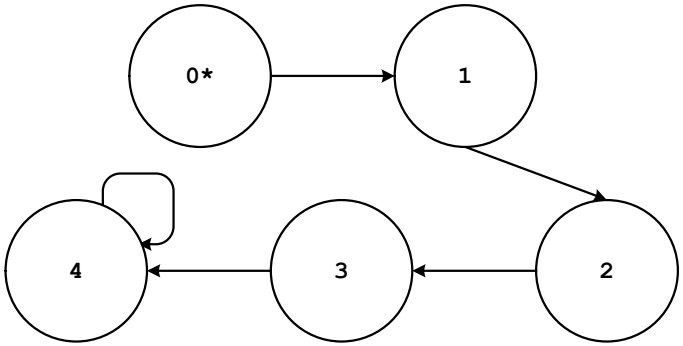
taskSelectSymbol: Coding




```
void taskSelectSymbol() {
  sym_ix++;
  if (sym_ix == 6) sym_ix = 2;
  tblink = 0;
}
```

State variable: sym_ix

taskDrawBlock: Coding



State variable: tblink

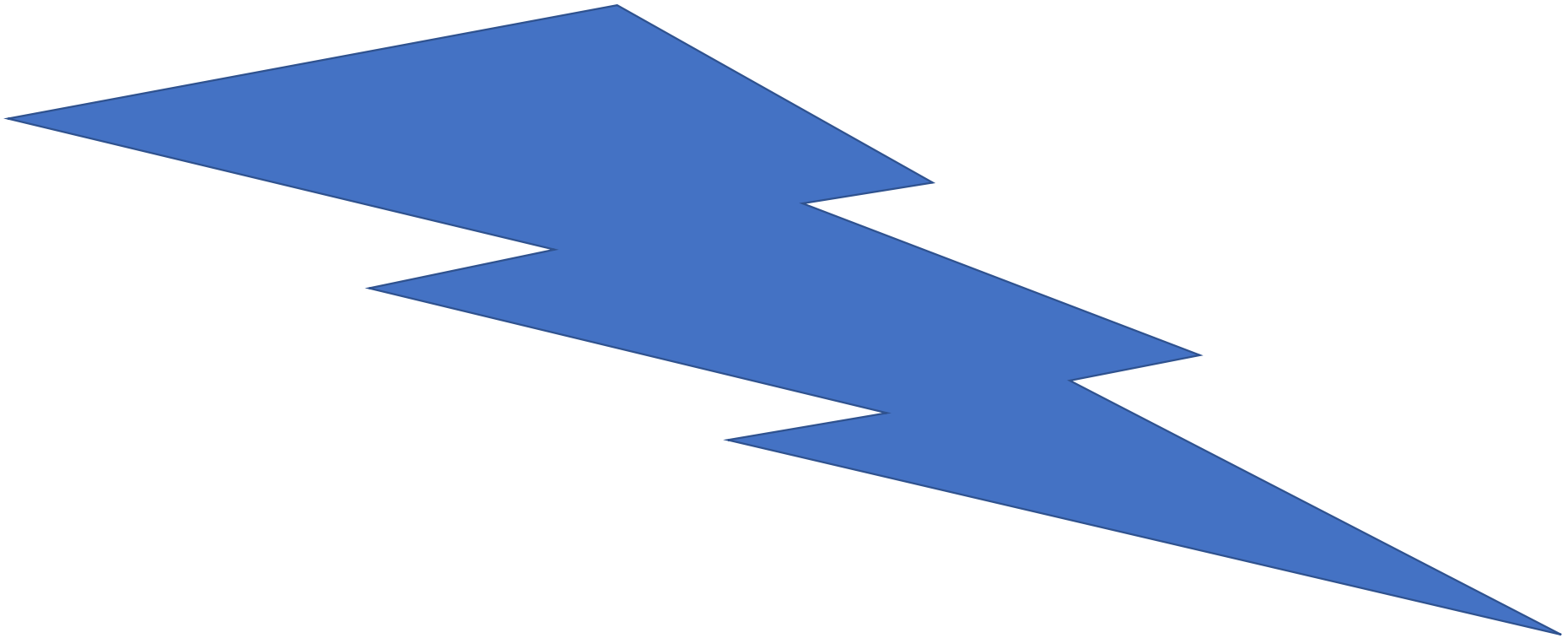
```
void taskDrawBlock() {
  int i;
  byte en_pattern;

  switch (tblink) {
    case 0:
      en_pattern = 0b00000000;
      break;
    case 1:
      en_pattern = 0b00000011;
      break;
    case 2:
      en_pattern = 0b00001111;
      break;
    case 3:
      en_pattern = 0b00111111;
      break;
    default:
      en_pattern = 0b11111111;
      break;
  }
  for (i = 0; i < 8; i++) {
    MAX7219_write_reg(REG_DIGIT(i), pattern[sym_ix][i] & en_pattern);
  }
  if (tblink < 4) tblink++;
}
```

State initializing

```
void setup() {
    delay(3000);
    // H/W initialize
    SPI.begin();
    SPI.setBitOrder(MSBFIRST);
    SPI.setClockDivider(SPI_CLOCK_DIV16);
    SPI.setDataMode(SPI_MODE0);
    pinMode(CS_PIN, OUTPUT);
    digitalWrite(CS_PIN, HIGH);
    MAX7219_init();
    // Variables and state initialize
    tick_last = millis();   //for sceduler
    rqtask = 0;
    rqCount = 0;
    preCount = 0;
    sym_ix = 2;
    tblink = 0;
    // End of setup
    MAX7219_write_reg(REG_SHUTDOWN, 0x01);
}
```

L2p5



L02p05: Another effect, design flow concept

Try to design yourself.

Use the given source code L02p05 as key.

Your final code is not necessary to be same as the key.

Design process should be focused.

- Requirement and function extraction
- Hardware
- Software modeling
- Coding
- Debugging