Embedded Systems (10)

- Will start at 15:10
- PDF of this slide is available via ScombZ

Hiroki Sato <i048219@shibaura-it.ac.jp>

15:10-16:50 on Wednesday

Targets At a Glance

- **What you will learn today**
  - Model-based development
  - State machine

- **Today's Project**
  - Using a matrix switch (keypad)

Design Patterns of Embedded Systems

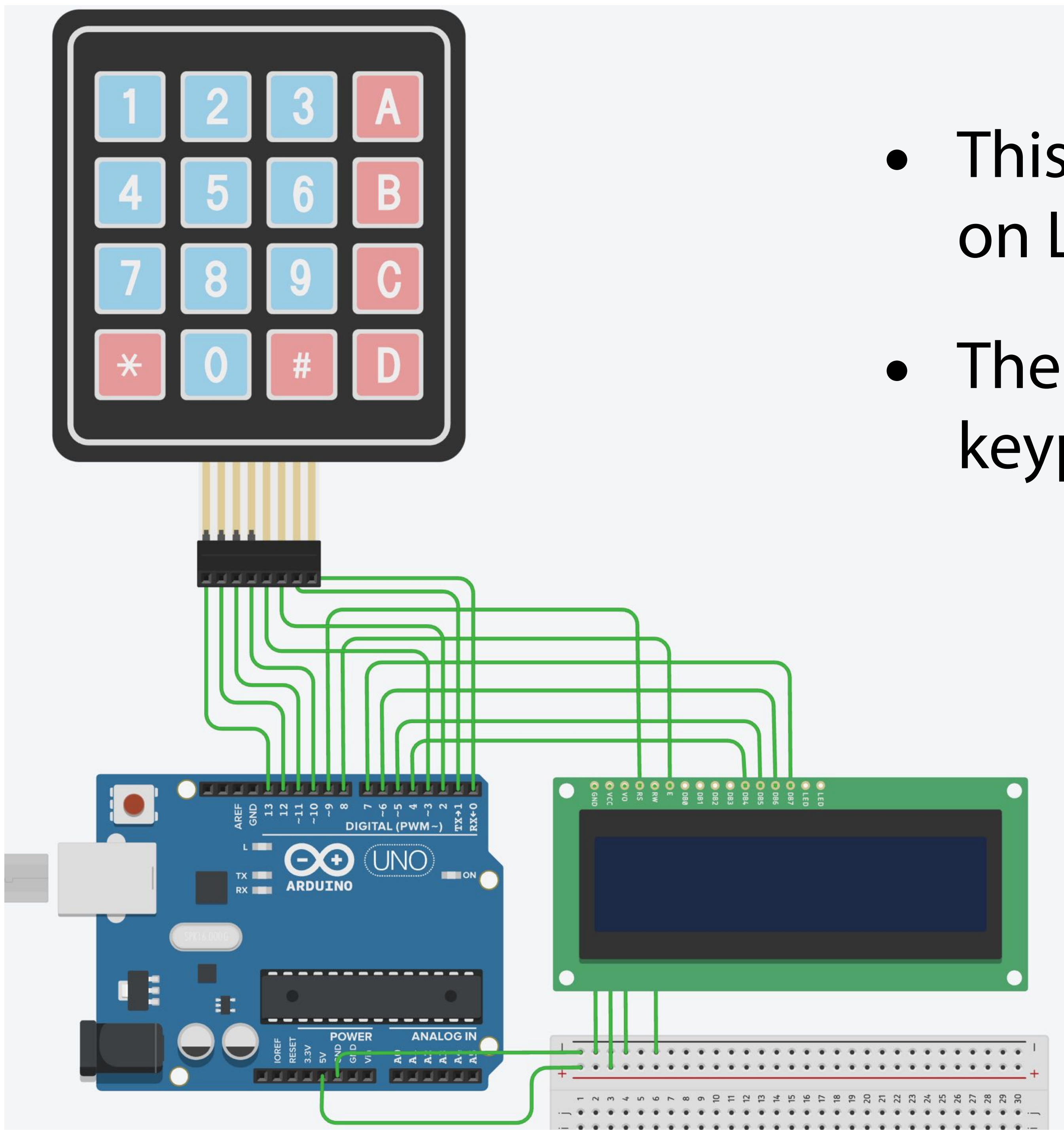
- **Information flow: input -> processing -> output**
- **The inputs are often "human-driven" on general-purpose computers.**
- **Programs for embedded systems tend to be "event-driven":**
    - loop() is an infinite loop to wait for events. Interrupt handlers are another entry points to accept events.
    - Then some processing is performed
    - Then the results will be sent to the output devices

Complexity and Reliability

- **Difficult to check if your program (and system) works as expected**
  - An event-driven program depends on events, reactions to them, etc.
- "System modeling" is important for developing a complex system.
  - Split the design into "expected behaviors" and "implementation (program)."

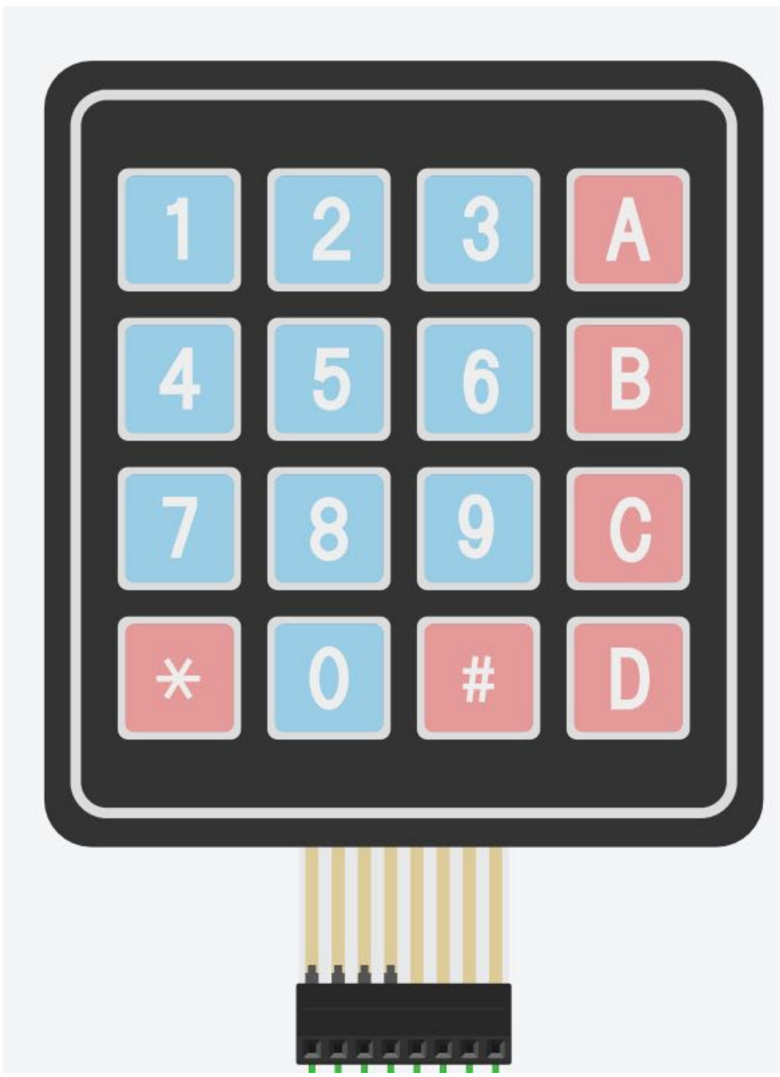
State Machine

Example: Matrix Switch (keypad)

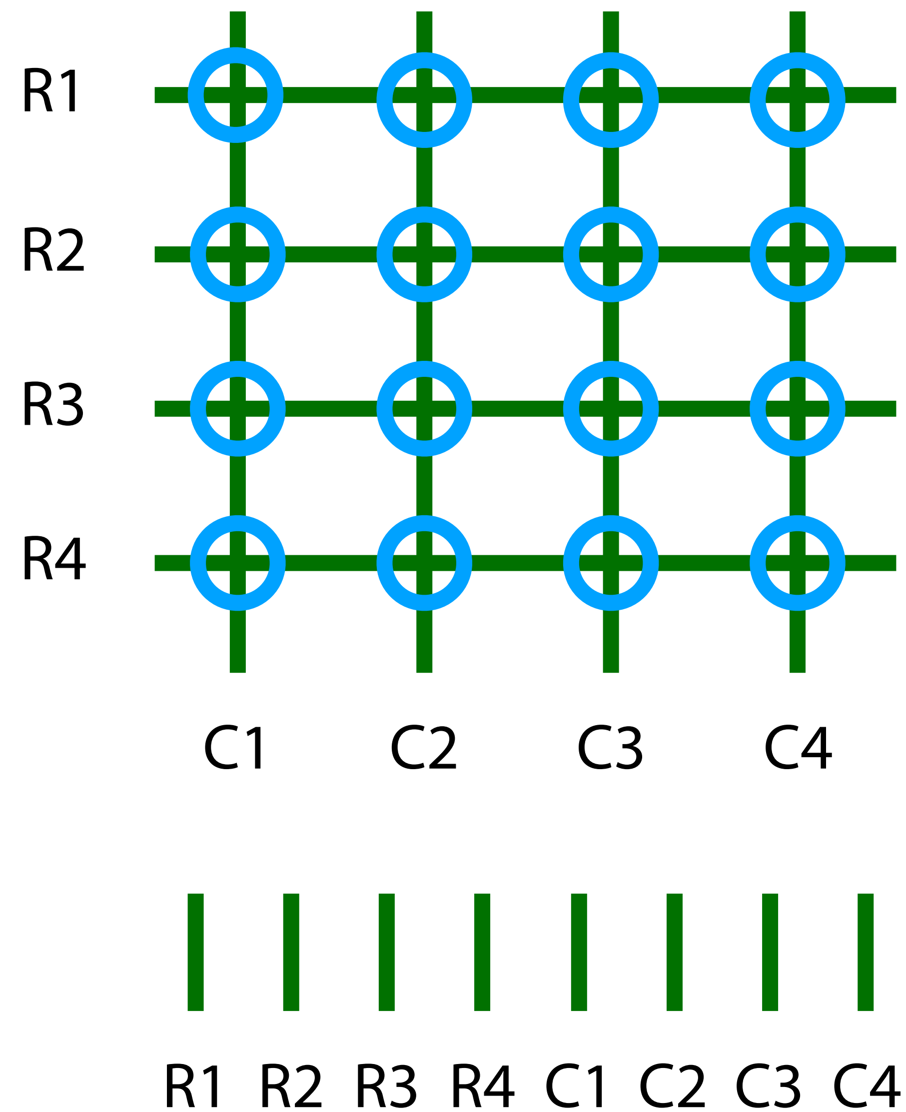


- This is a system that shows a character on LCD when pressing a button.
- The characters are ones on the keypad.

Example: Matrix Switch (keypad)



8 pins



8 pins

- The complexity of handling a keypad
- Needs to check 8 wires every time.

```
const char row[] = {13, 12, 11, 10};
const char column[] = {3, 2, 1, 0};
const char key[][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

char
scan_keypad()
{

  for (int i = 0; i < sizeof(row); i++)
    digitalWrite(row[i], HIGH);

  for (int i = 0; i < sizeof(row); i++) {
    digitalWrite(row[i], LOW);

    for (int j = 0; j < sizeof(column); j++) {
      if (digitalRead(column[j]) == LOW)
        return key[i][j];
    }
  }
  return (0); /* No pressed key */
}
```

Example: Matrix Switch (keypad)

- Complexity of handling a keypad (cont'd)
  - **Two events of pressing and releasing must be detected**
    - Checking only a press is not enough
    - Both checking pressing or releasing requires scanning of the 8 wires
- **There will be a lot of if-then-else statements that calls scan_keypad().**
  - To deal with this kind of complexity, state machine is often used.

State Machine

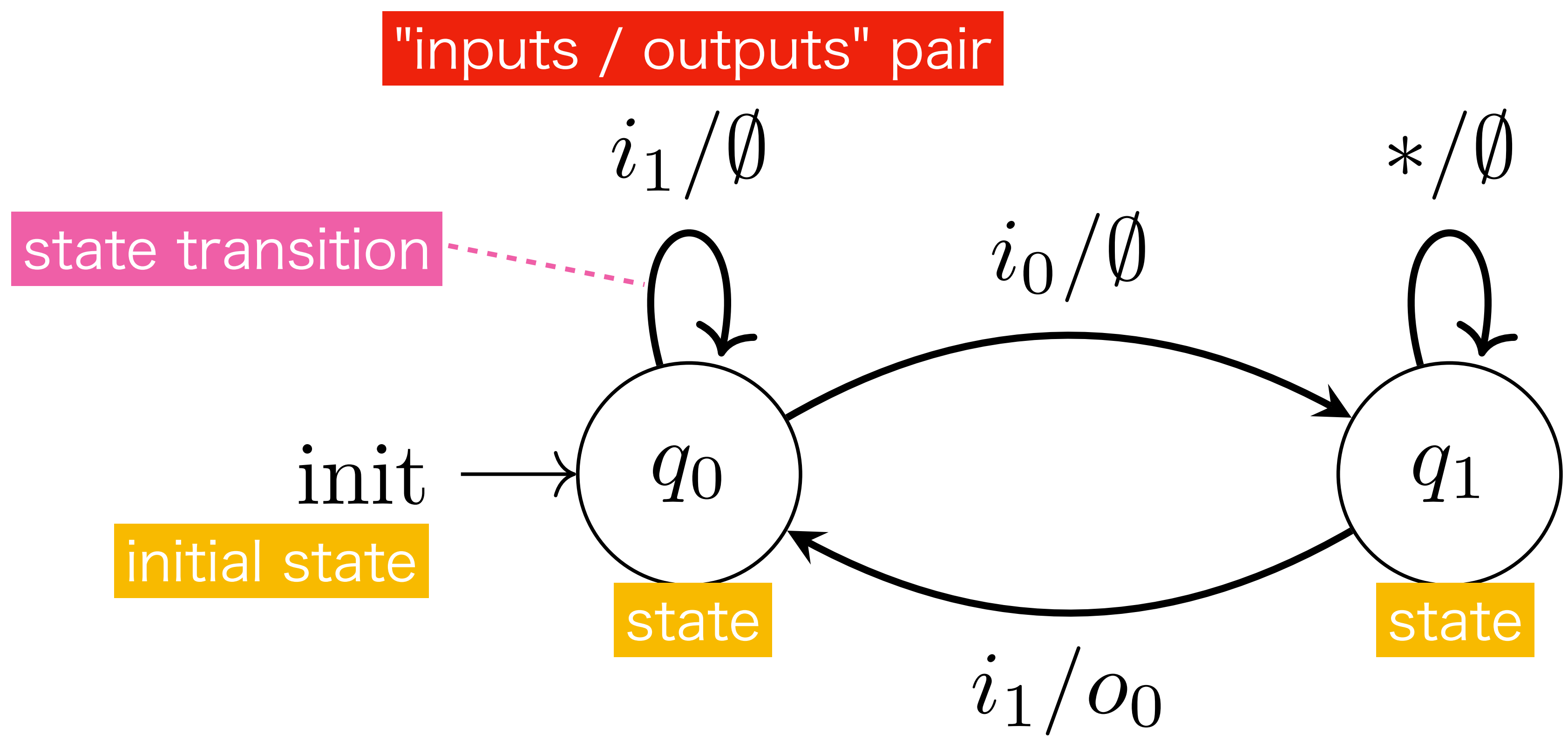
- An event-driven system can be modeled by using FSM (finite state machine or finite automaton).
- The actors are "inputs", "states", "outputs"

$$\begin{aligned}
\mathrm{SYSTEM} &= \{S, \Sigma, \Lambda, T, G, s\} \\
S &= \text{states} \\
\Sigma &= \text{inputs} \\
\Lambda &= \text{outputs} \\
T &= \text{transition function} : S \times \Sigma \to S \\
G &= \text{output function} : S \times \Sigma \to \Lambda \\
s &= \text{initial state}
\end{aligned}$$

State Machine

- An event-driven system can be modeled by using FSM (finite state machine or finite automaton).
- The actors are "inputs", "states", "outputs"
  - A system has a set of inputs, states, and outputs.
  - "state" represents the current state of the system. It accepts "inputs", and then sets the corresponding "outputs" and goes to the next state.
  - Consider "input -> processing -> output" information flow

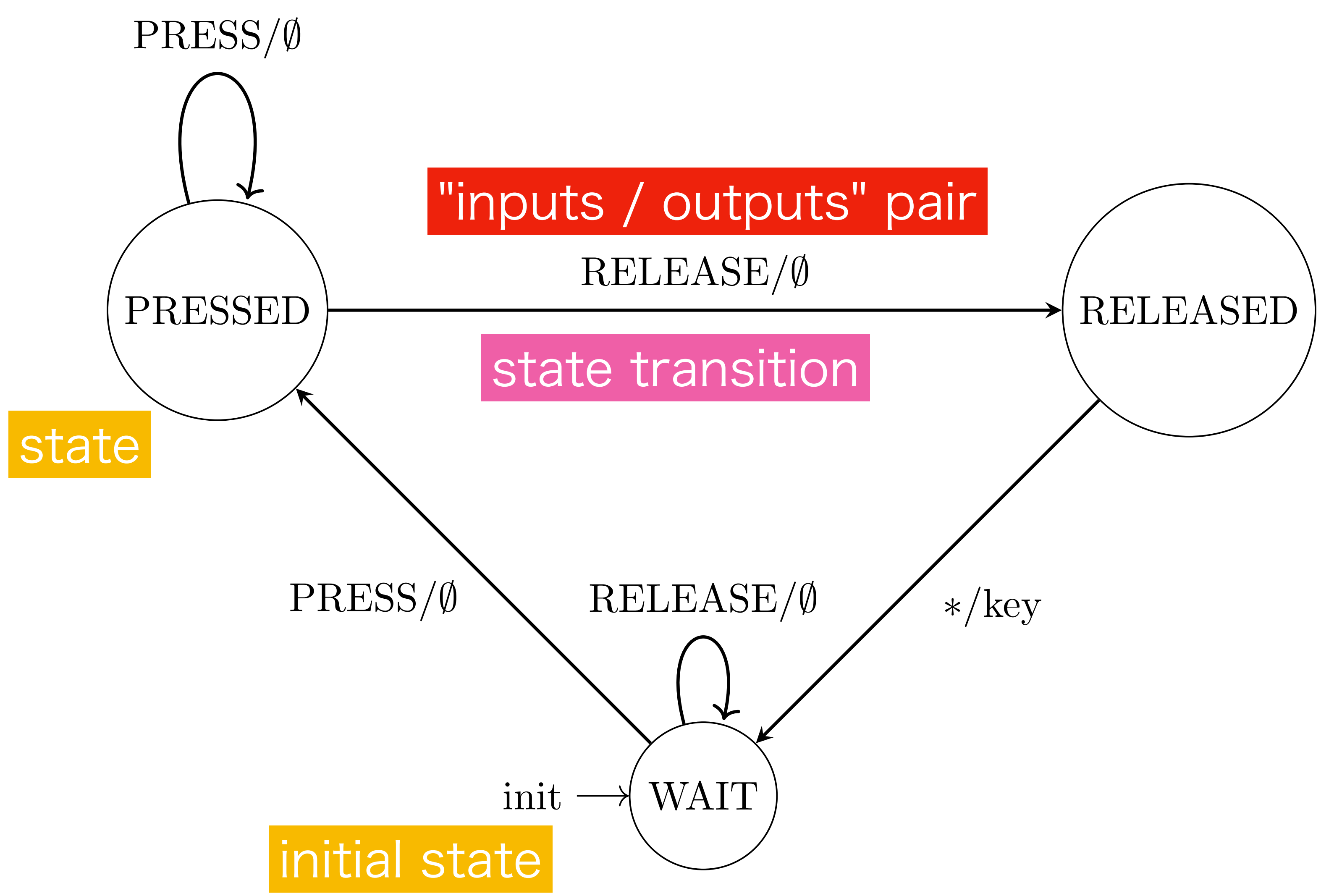
State Transition Diagram



$$\{\{q_0, q_1\}, \{i_0, i_1\}, \{o_0\}, T, G, s\}$$

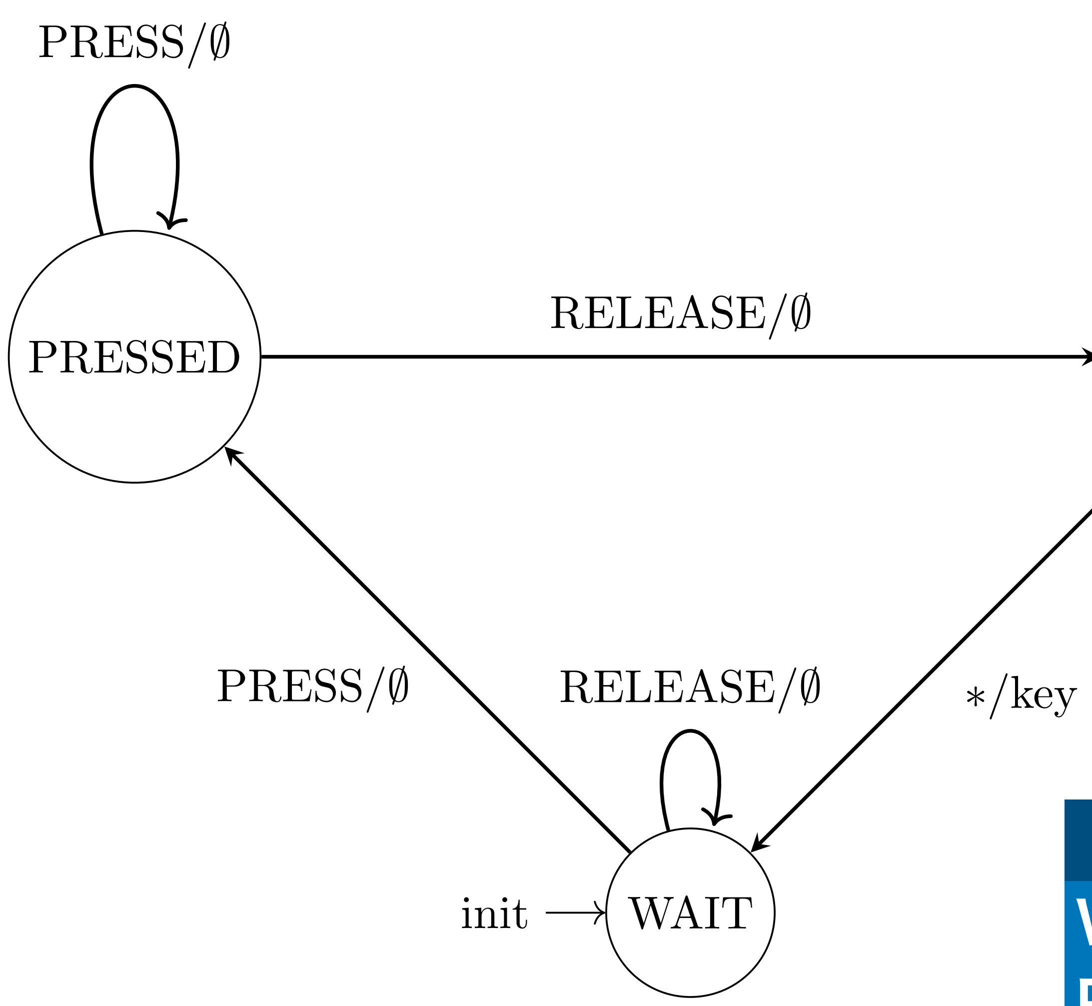
- Every state must have edges corresponding to # of inputs.

State Transition Diagram



$$\{\{\mathrm{WAIT}, \mathrm{PRESSED}, \mathrm{RELEASED}\}, \{\mathrm{PRESS}, \mathrm{RELEASE}\}, \{\mathrm{key}\}, T, G, s\}$$

State Transition Table



S\Σ	PRESS	RELEASE
WAIT	PRESSED/ ϕ	WAIT/ ϕ
PRESSED	PRESSED/ ϕ	RELEASED/ ϕ
RELEASED	WAIT/key	WAIT/key

S/ Λ

```c
/* State Machine */
enum input_t {
  I_RELEASE,
  I_PRESS,
  I_MAX
};

enum state_t {
  S_WAIT,
  S_PRESSED,
  S_RELEASED,
  S_MAX
};

struct state_t {
  enum state_set s;
  void (*output)(const char);
};

struct state_t s_next[S_MAX][I_MAX] = {
  [S_WAIT] = {
    [I_RELEASE] = { S_WAIT, do_nothing },
    [I_PRESS] = { S_PRESSED, do_nothing }
  },
  [S_PRESSED] = {
    [I_RELEASE] = { S_RELEASED, do_nothing },
    [I_PRESS] = { S_PRESSED, do_nothing }
  },
  [S_RELEASED] = {
    [I_RELEASE] = { S_WAIT, update_lcd1 },
    [I_PRESS] = { S_WAIT, update_lcd1 }
  }
};
```

the input set

the internal state set

next state

output function

S\ Σ	PRESS	RELEASE
WAIT	PRESSED/ ϕ	WAIT/ ϕ
PRESSED	PRESSED/ ϕ	RELEASED/ ϕ
RELEASED	WAIT/key	WAIT/key

- A state transition table can be directly implemented as an array.
- This guarantees that every possible transition are covered on the system.

```
char k0; /* pressed key in S_PRESSED */

struct input_t
get_input(void)
{
  struct input_t i;

  i.key = scan_keypad();
  if (i.key == 0) {
    i.s = I_RELEASE;
    i.key = k0;
  } else {
    i.s = I_PRESS;
    k0 = i.key;
  }
  return (i);
}
```

input function

S\Σ	PRESS	RELEASE
WAIT	PRESSED/ ϕ	WAIT/ ϕ
PRESSED	PRESSED/ ϕ	RELEASED/ ϕ
RELEASED	WAIT/key	WAIT/key

```
/* Current state */
struct state_t state = { S_WAIT, NULL };

void
loop() {
  struct input_t i;

  i = get_input();
  state = s_next[state.s][i.s];
  (*state.output)(i.key);
  delay(50);
}
```

state transition

- The inputs and the outputs can be handled in a consistent way
- You can add more functionality as "state" without losing the consistency.

```
#include <LiquidCrystal.h>
#include <stdio.h>

/* State Machine */
enum input_set {
  I_RELEASE,
  I_PRESS,
  I_MAX
};
struct input_t {
  enum input_set s;
  char key;
};

enum state_set {
  S_WAIT,
  S_PRESSED,
  S_RELEASED,
  S_MAX
};
struct state_t {
  enum state_set s;
  void (*output)(const char);
};

struct state_t s_next[S_MAX][I_MAX] = {
  [S_WAIT] = {
    [I_RELEASE] = { S_WAIT, do_nothing },
    [I_PRESS] = { S_PRESSED, do_nothing }
  },
  [S_PRESSED] = {
    [I_RELEASE] = { S_RELEASED, do_nothing },
    [I_PRESS] = { S_PRESSED, do_nothing }
  },
  [S_RELEASED] = {
    [I_RELEASE] = { S_WAIT, update_lcd1 },
    [I_PRESS] = { S_WAIT, update_lcd1 }
  }
};
```

```
/* Keypad */
char k0; /* pressed key in S_PRESSED */
/* NOTE: do not use "Serial" because it uses pin0
and pin1 */
const char row[] = {13, 12, 11, 10};
const char column[] = {3, 2, 1, 0};
const char key[][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

char
scan_keypad()
{

  for (int i = 0; i < sizeof(row); i++)
    digitalWrite(row[i], HIGH);
  for (int i = 0; i < sizeof(row); i++) {
    digitalWrite(row[i], LOW);

    for (int j = 0; j < sizeof(column); j++) {
      if (digitalRead(column[j]) == LOW)
        return key[i][j];
    }
  }
  return (0); /* No pressed key */
}
```

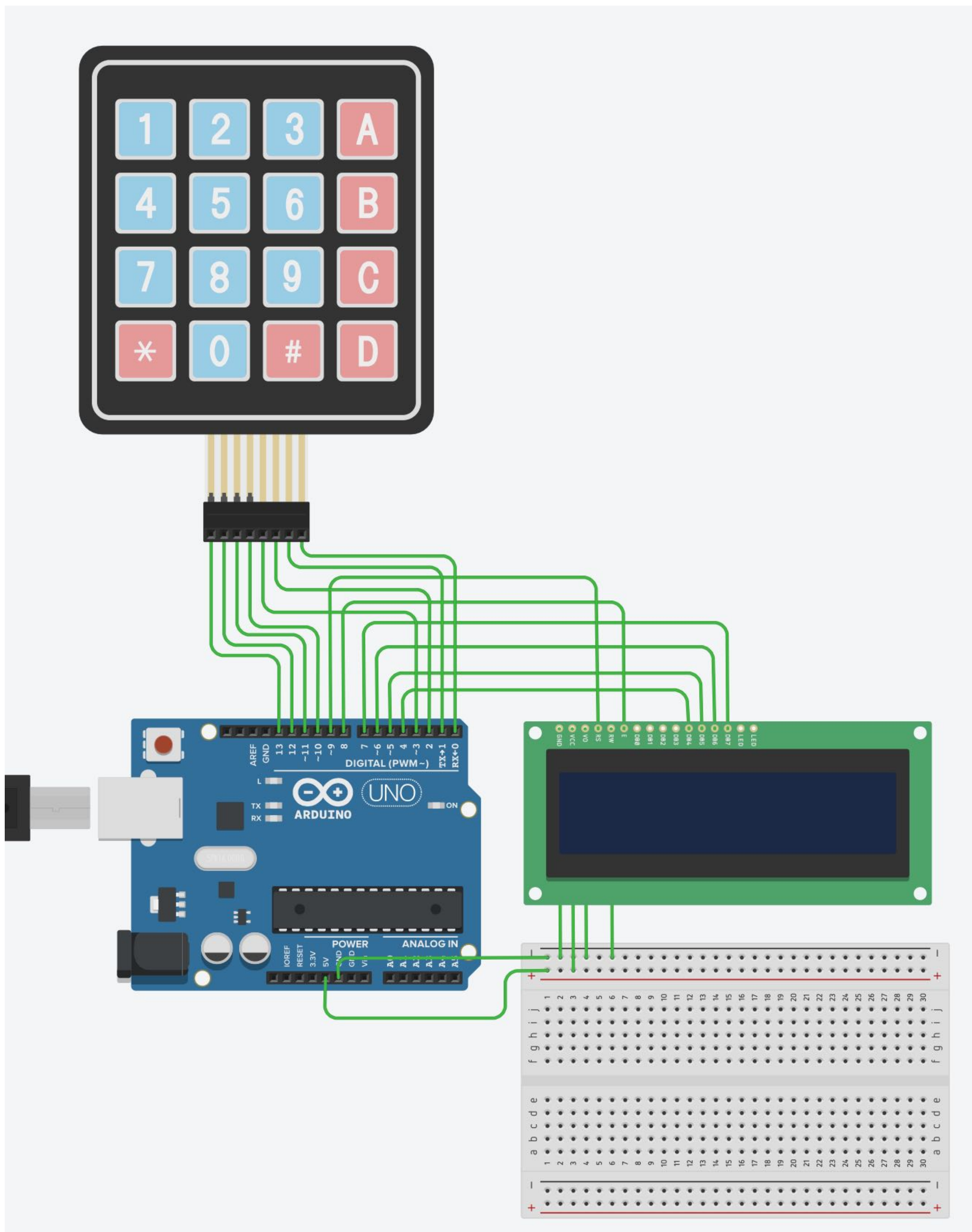
S\Σ	PRESS	RELEASE
WAIT	PRESSED/ϕ	WAIT/ϕ
PRESSED	PRESSED/ϕ	RELEASED/ϕ
RELEASED	WAIT/key	WAIT/key

```
/* LCD */
#define   RS 9
#define   EN 8
#define   DB4 4
#define   DB5 5
#define   DB6 6
#define   DB7 7
LiquidCrystal lcd(RS, EN, DB4, DB5, DB6,
DB7);

char line0[17] = "hello, world";
char line1[17];
char pos1; /* cursor */

void
do_nothing(const char c)
{
}

void
update_lcd1(const char c)
{

  lcd.setCursor(0, 1);
  /* Clear */
  if (pos1 == 0) {
    memset(line1, ' ', sizeof(line1));
  }
  line1[pos1] = c;
  /* Termination */
  line1[sizeof(line1) - 1] = '\0';

  pos1 = (pos1 + 1) % (sizeof(line1) - 1);
  lcd.print(line1);
}
```

```
struct input_t
get_input(void)
{
  struct input_t i;

  i.key = scan_keypad();
  if (i.key == 0) {
    i.s = I_RELEASE;
    i.key = k0;
  } else {
    i.s = I_PRESS;
    k0 = i.key;
  }
  return (i);
}

void
setup() {
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(line0);

  for (int i = 0; i < sizeof(row); i++)
    pinMode(row[i], OUTPUT);
  for (int j = 0; j < sizeof(column); j++)
    pinMode(column[j], INPUT_PULLUP);
}

/* Current state */
struct state_t state = { S_WAIT, NULL };

void
loop() {
  struct input_t i;

  i = get_input();
  state = s_next[state.s][i.s];
  (*state.output)(i.key);
  delay(50);
}
```

Demo

Time for Your Project



- Try to implement a system that shows pressed keys as characters on LCD, **a) with a state machine and then b) without a state machine.**
- Note that a) is already shown in the previous pages. Try to understand it.

Time for Your Project

- If you finish a) and b), try implementing a c)keypad+LCD+blinker that has a blinking "＊" on the first line in addition to the original keypad+LCD function. The blinker can be added using the timer interrupts.
- if you finish keypad+LCD+blinker, try to implement a d)calculator based on it:
    - 'B' for addition, 'C' for subtraction, 'D' to get the result, 'A' to start over.
    - Design the state transition diagram first, and then implement it.

Conclusions

- **Next week**:
  - Example answer of 8-C
  - Model-based development (continued)
  - Operating systems and examples