Embedded Systems (11)

- 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 (continued)
  - Preparation for the end-term project
  - December 20 will be the last class this year. The remaining technical topics will be covered on that day.

- **Today's Project**
  - Using multiple state machines

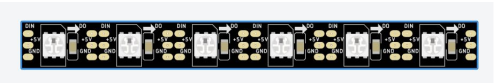
Regarding 8-C

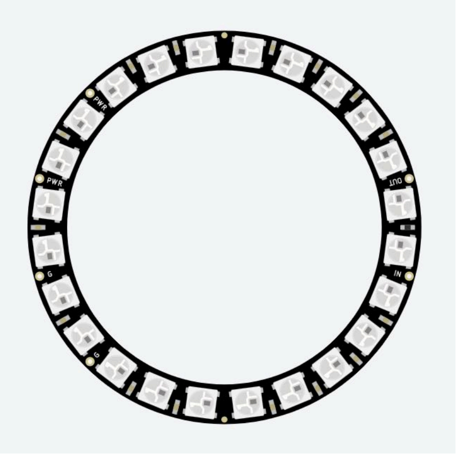
- **You will receive review results via email by the next Sunday at the latest**
  - If your design needs to be fixed, it will be pointed out.
  - Your questions will be answered in the reply, and if it is worth sharing I will also explain it in the next week.
  - No re-submission is required if yours works fine. If not, resubmit a revised version. Read instructions in the email carefully.

End Term Project

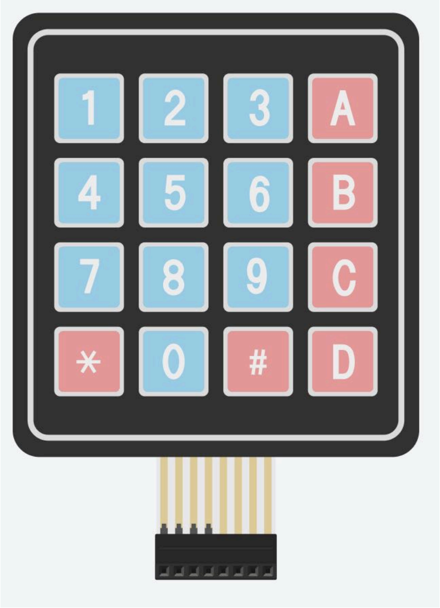
End Term Project

- **Implement your own embedded system that**
  - has the three components (input, processor, and output) for information processing,
  - does communication between two or more boards,
  - uses the model-based development methodology.
  - While you can use any available components, no library for communication is allowed for exercise purpose.
- **Examples (not limited to):**
  - Simple games such as hit-and-blow or roulette
  - Calculator using keypads and LCD
  - Level meter of the output of sensors





LED array



Key pad

End Term Project

- **Schedule**

Dec 13	Dec 20	Jan 10	Jan 17	Jan 20
Day 1	Day 2	Day 3	Day 4	Submission due

Start to plan

- **You must write and submit a report (in PDF) and the design on TinkerCAD by 23:59 on January 20th (JST).**
- **You will receive a real development board on December 20th. If your design works on it, it will gives extra score.**
- **Evaluation**
  - 80% for the end-term project and 20% for the exercises.
- **Questions**
  - Ask the teaching assistant during class hours or email the instructor.

End Term Project

- **Chapter formation of the report:**

  (1)Cover page (title, your name, student ID, and project name in TinkerCAD)

  (2)The objective of your system

  (3)Instructions (how to use it)

  (4)Hardware and software structure (how did you design it)

  (5)Reference list (URL, textbook, etc.)

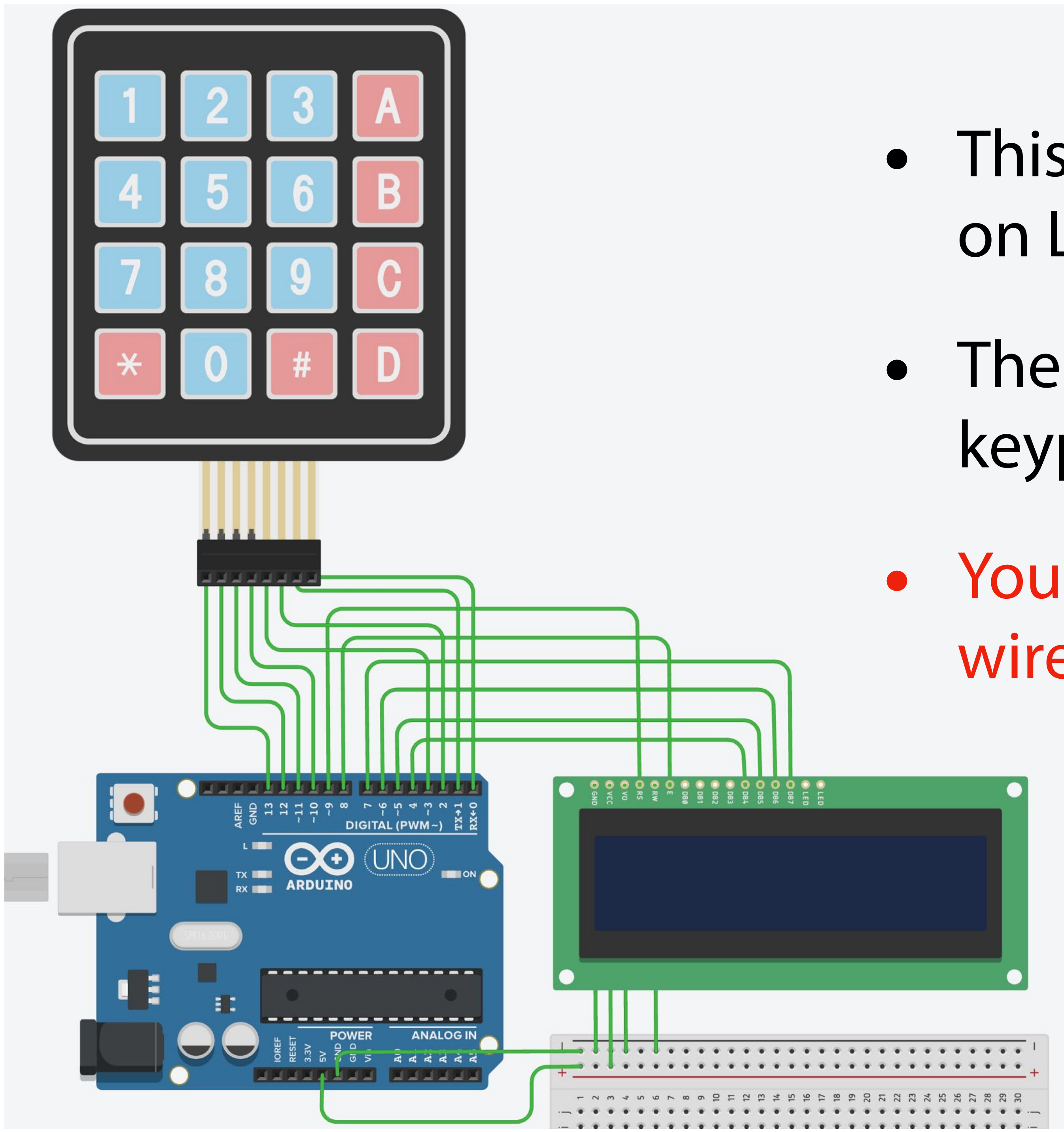
- **In (2), explain what the motivation for your design was.**
- **The (4) must include what you planned out and all your program listings with your comments.**
- **If you could not complete what you expected, the report should describe what you did.**
- **The report must be in English or Japanese.**

End Term Project

- **Evaluation Criteria**
    - **Originality**. Discussing with your friends and/or doing a research on Internet is a good thing, but do not copy. And you must provide a list all of the references.
    - **Source code you wrote by yourself**. Using libraries except for communication is okay, but you will get more scores for things you wrote by yourself.
    - **How well you described your design in the report**. Consider a "real-life situation" and how your system improves it.
- Designs not satisfying the mandatory requirements will get lower scores.

State Machine

Example: Matrix Switch (keypad)



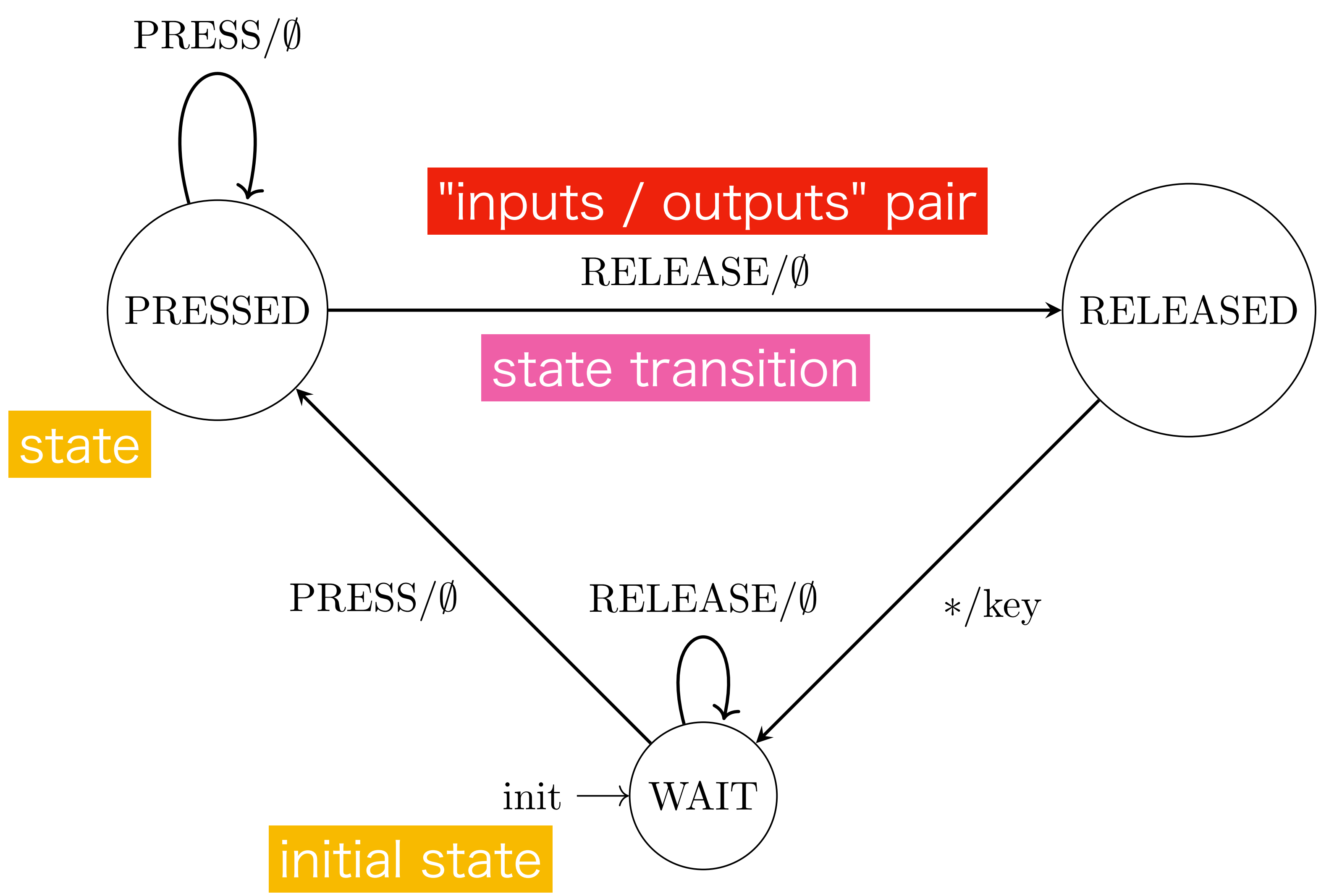
- This is a system that shows a character on LCD when pressing a button.
- The characters are ones on the keypad.
- You cannot use interrupts because 8 wires must be scanned

State Machine

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

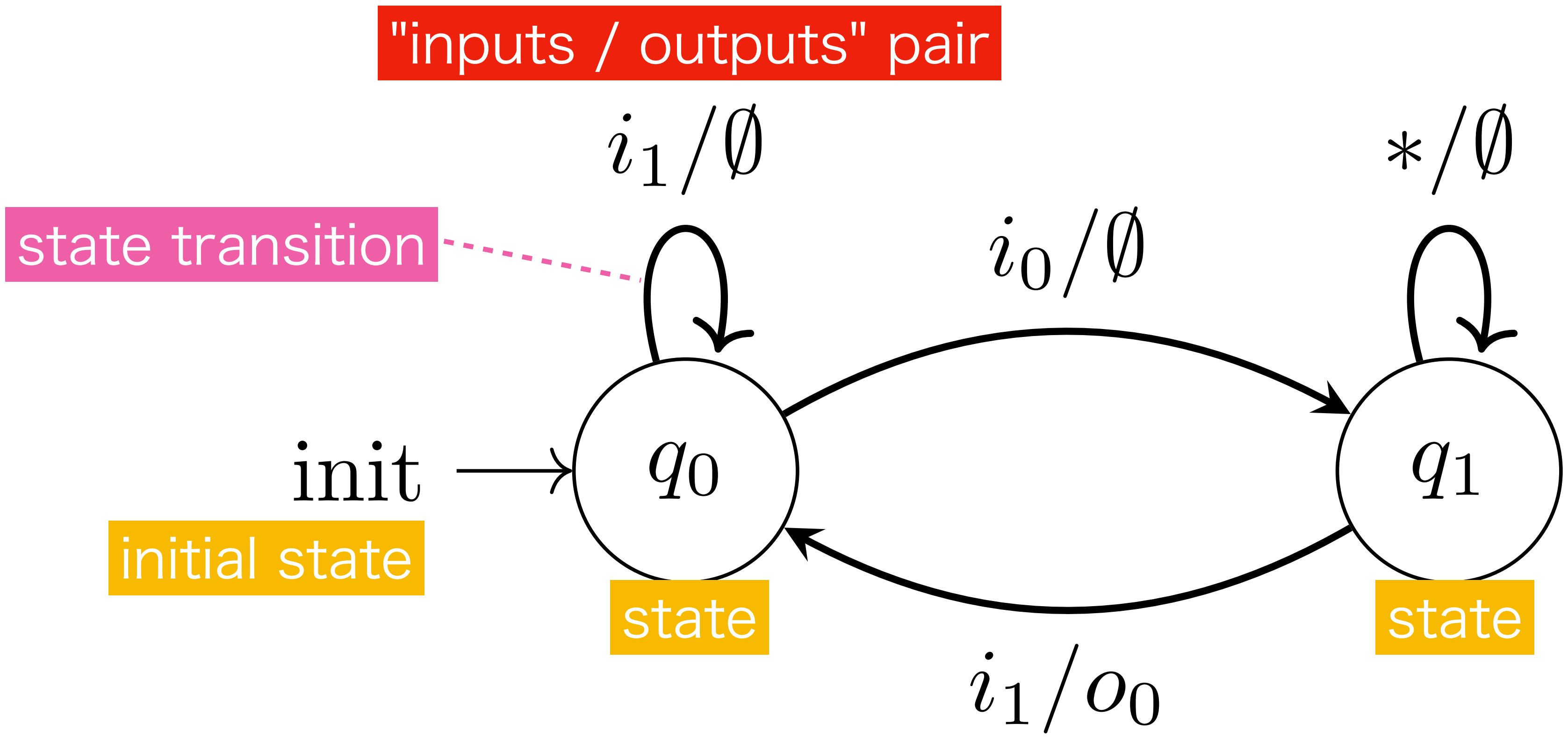
$$
\begin{aligned}
\mathrm{SYSTEM} &= \{S, \Sigma, \Lambda, T, G, s\} \\
S &= \mathrm{states} \\
\Sigma &= \mathrm{inputs} \\
\Lambda &= \mathrm{outputs} \\
T &= \mathrm{transition\ function} : S \times \Sigma \rightarrow S \\
G &= \mathrm{output\ function} : S \times \Sigma \rightarrow \Lambda \\
s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

State Transition Diagram



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

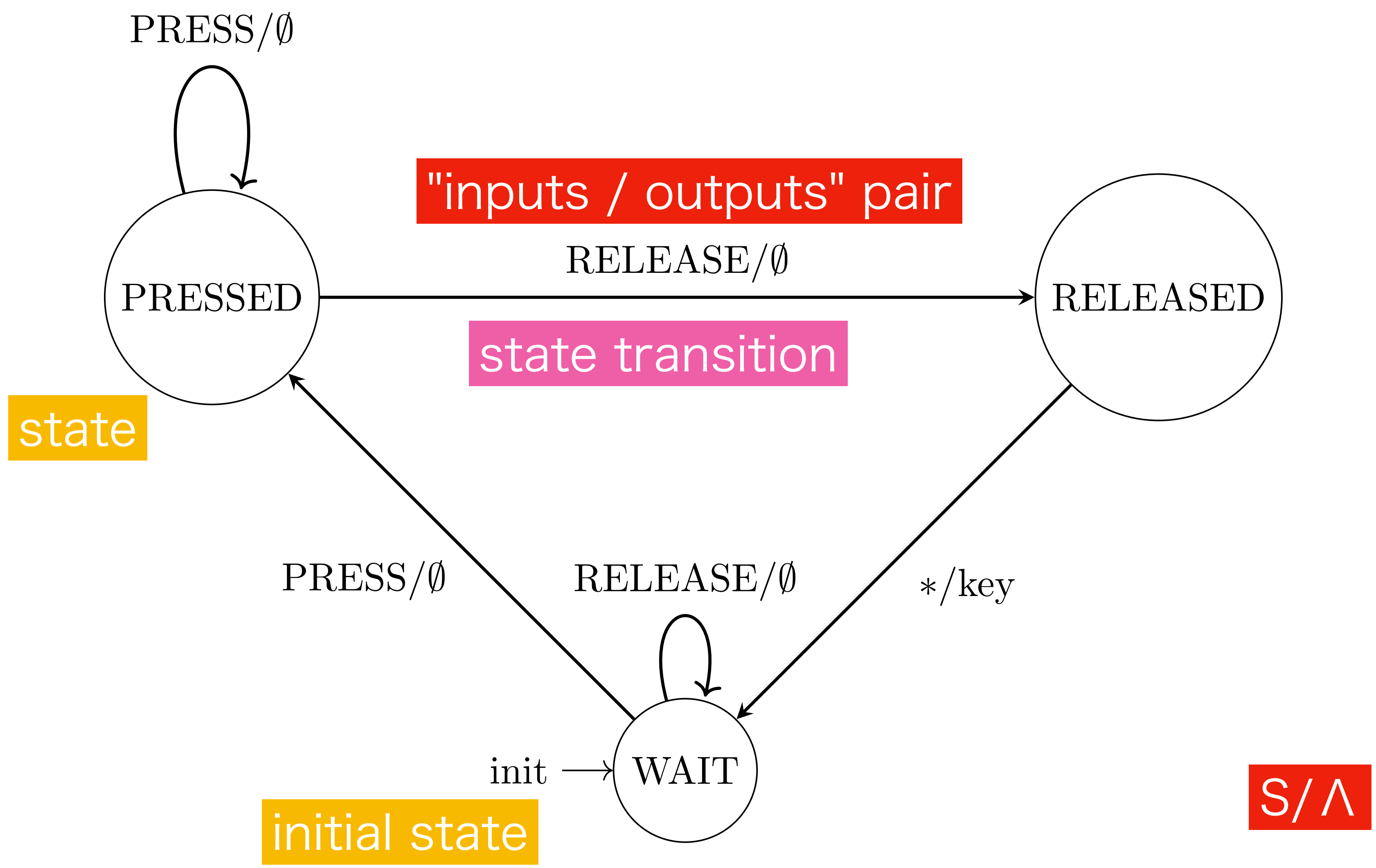
State Transition Diagram



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

- 1) a function to get an input Σ.
- 2) a function $T(\Sigma, S_n)$ that takes an input Σ and the current state S_n and generates the next state S_{n+1}
- 3) a functions $G(\Sigma, S_n)$ to generate output Λ

State Transition Table



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

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

the input set

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

the internal state set

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

next state Sn+1

output function G(Σ)

```
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 }
  }
};
```

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 transitions 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);
}

/* 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);
}
```

input function to generate Σ

transition function T(Sn, Σ) (implemented as an array)

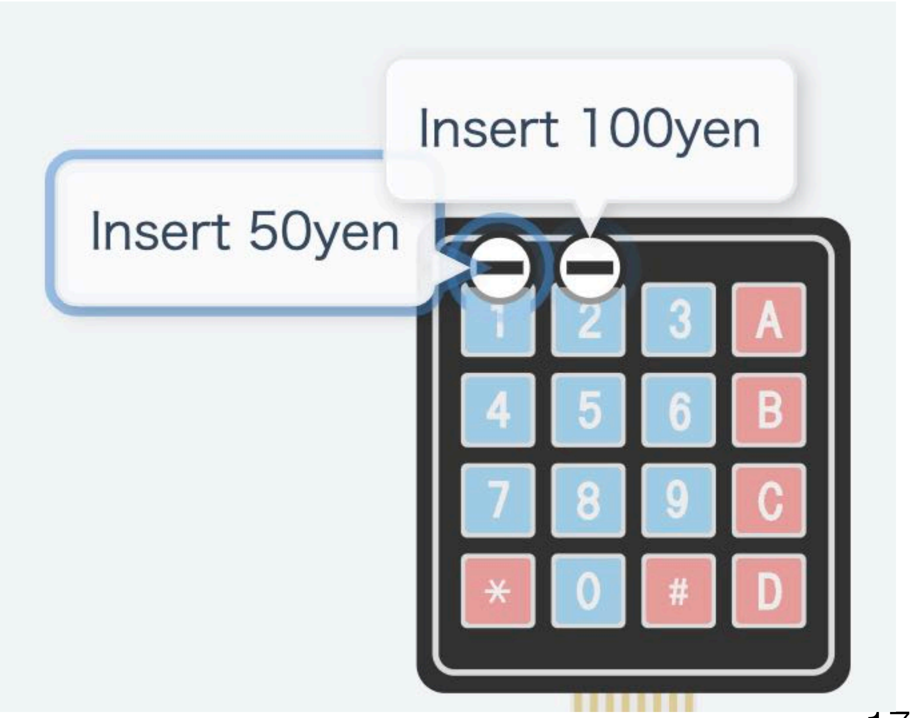
S\Σ	PRESS	RELEASE
WAIT	PRESSED/ϕ	WAIT/ϕ
PRESSED	PRESSED/ϕ	RELEASED/ϕ
RELEASED	WAIT/key	WAIT/key

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

Multiple State Machines

- **Example: Vending machine**
  - This accepts 50yen and 100yen coins. Keypad is used to simulate insertion of coins.
  - A can comes out when the machine has 150yen or more. Changes also come out. This is simulated using LCD.

- Input set = { φ, 50yen, 100yen }
- output set = { φ, a can, a can + 50 yen }
- state set = { WAITCOIN, HAS50, HAS100, RELEASE }



```
/* 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);
}
```

Single state machine

```
/* Current state */
struct state_t k_state = { SK_WAIT, NULL };
struct state_t v_state = { SV_WAITCOIN, NULL };

void
loop() {
  struct k_input_t ki;
  struct v_input_t vi;

  ki = get_k_input();
  k_state = k_s_next[k_state.s][ki.s];
  vi = (*k_state.output)(ki.key);

  v_state = v_s_next[v_state.s][vi.s];
  (*v_state.output)(vi.value);
  delay(50);
}
```

Multiple state machines

```
/* 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);
}
```

Single state machine

```
/* Current state */
struct state_t k_state = { SK_WAIT, NULL };
struct state_t v_state = { SV_WAITCOIN, NULL };

void
loop() {
  struct k_input_t ki;
  struct v_input_t vi;

  ki = get_k_input();
  k_state = k_s_next[k_state.s][ki.s];
  vi = (*k_state.output)(ki.key);

  v_state = v_s_next[v_state.s][vi.s];
  (*v_state.output)(vi.value);
  delay(50);
}
```

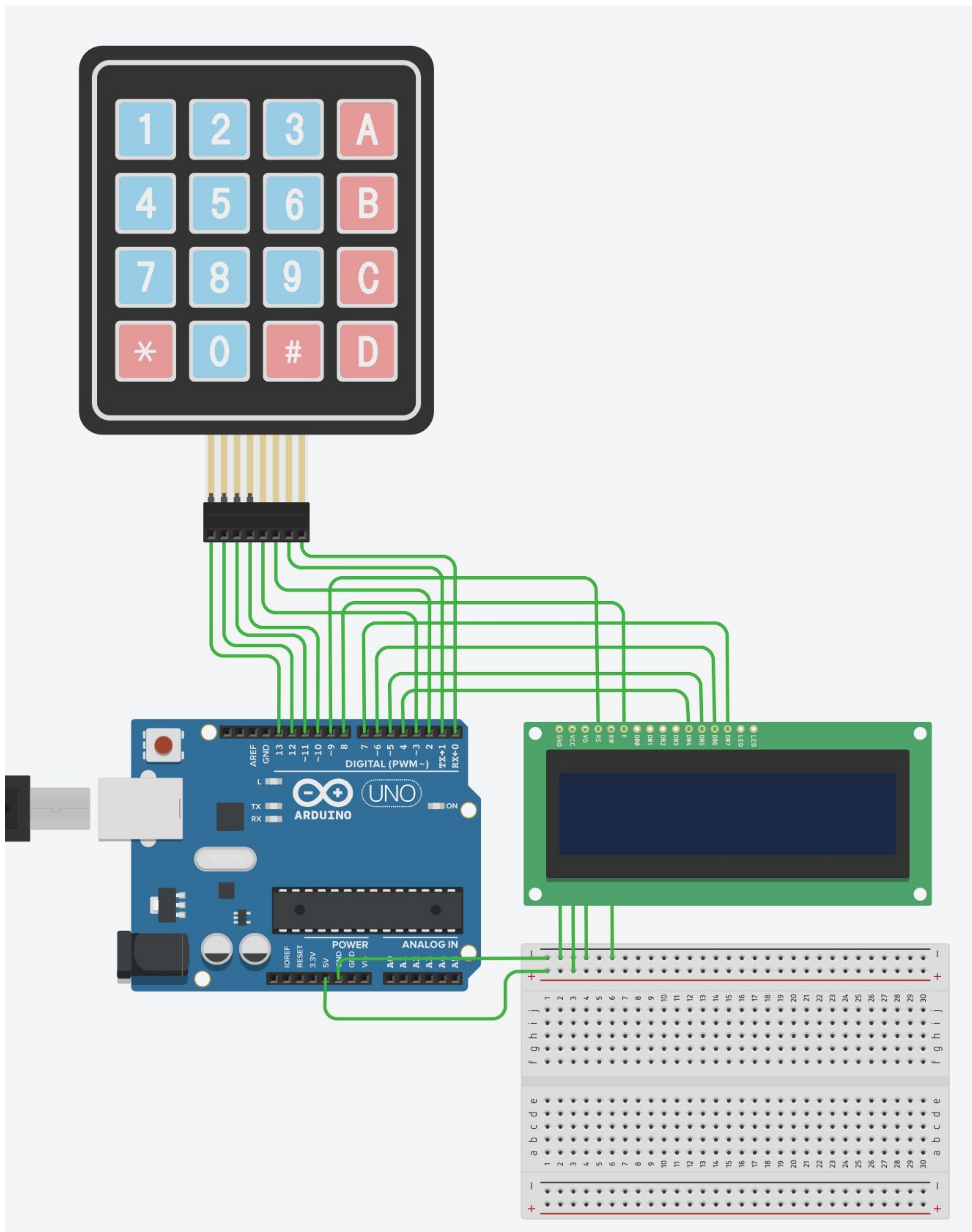
- k_state and v_state: two states must be maintained

- k_state.output() is the input function of v_state

Multiple State Machines

- **A state machine is...**
  - A single set of the input-process-output design pattern.
  - You can have multiple instances in the same system.
- **Design Considerations**
  - A single big table v.s. multiple small tables for state transitions:
    - The size will be {Σ} x {S}. It can rapidly increate in the O^2 order.
    - Splitting it into small tables may or may not simplify your model.
  - No not forget to check every possible combination of inputs and outputs. Your system must handle all of them for reliability.

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.
- Try a vending machine using multiple state machines.