Embedded Systems (11)

- Will start at 15:10
- Use "your student ID and your name" as your name on Zoom
- Mute your microphone during not speaking
- PDF of this slide is available via Scomb

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

15:10-16:50 on Wednesday

Targets At a Glance

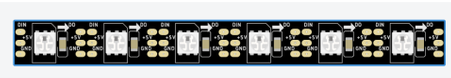
- **What you will learn today**
  - Example answer of the DESIGN-8C
  - Model-based development (continued)
  - Preparation for the end-term project

- **Today's Project**
  - Using a matrix switch (keypad)
  - Plan your end-term project

- Note: you can download source files of the example answers via Scomb

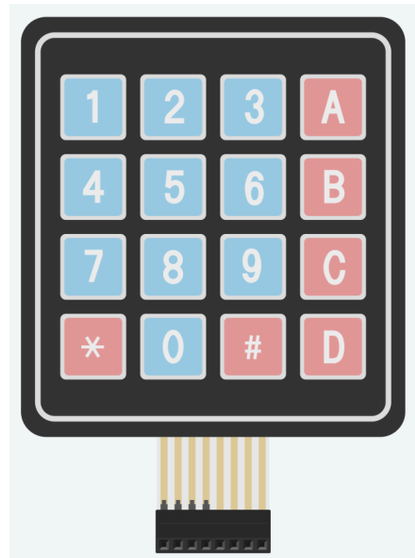
End Term Project

End Term Project



- **Implement your own embedded system which**
  - has the three components for some information processing,
  - does communication between two or more boards,
  - uses the model-based development methodology,
  - You can use any available components and libraries
- **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 14	Dec 21	Jan 11	Jan 18	Jan 22
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 22nd (JST).**

- **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

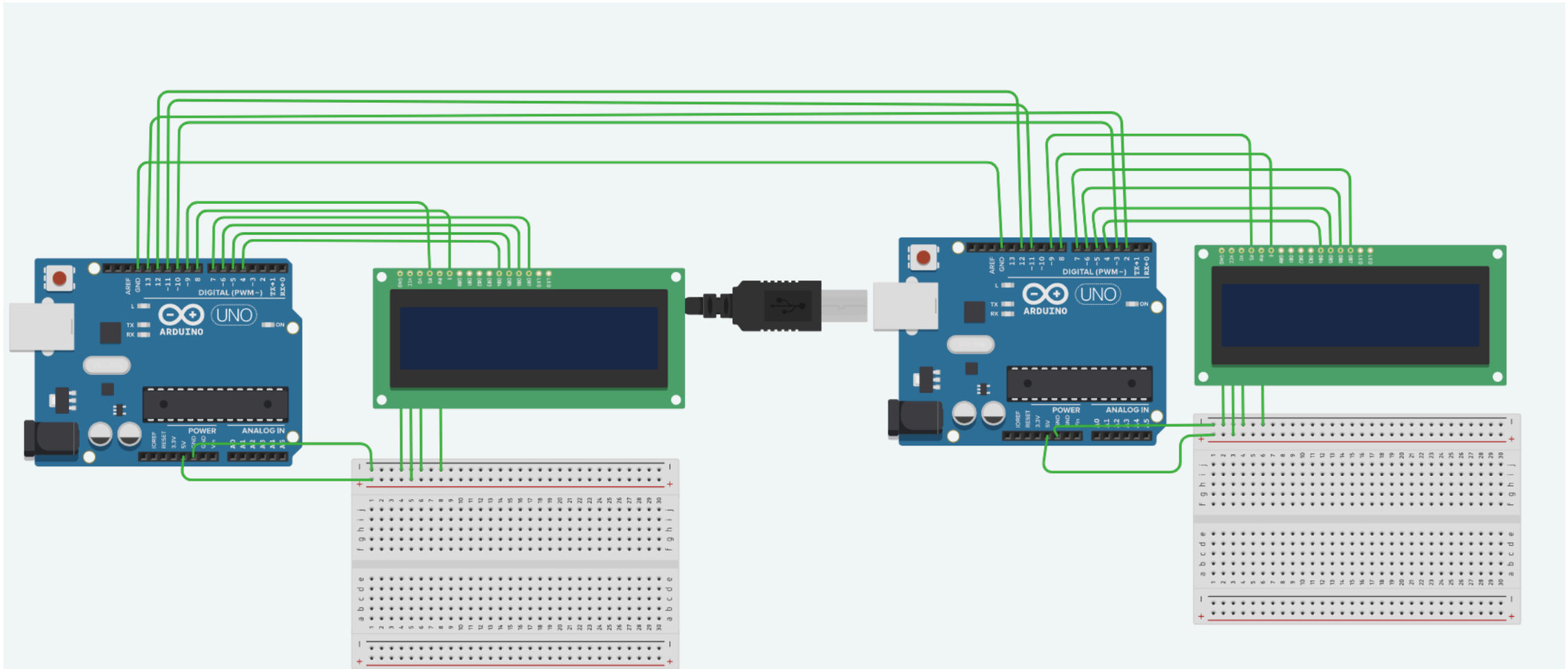
  (3)Instructions (how to use it)

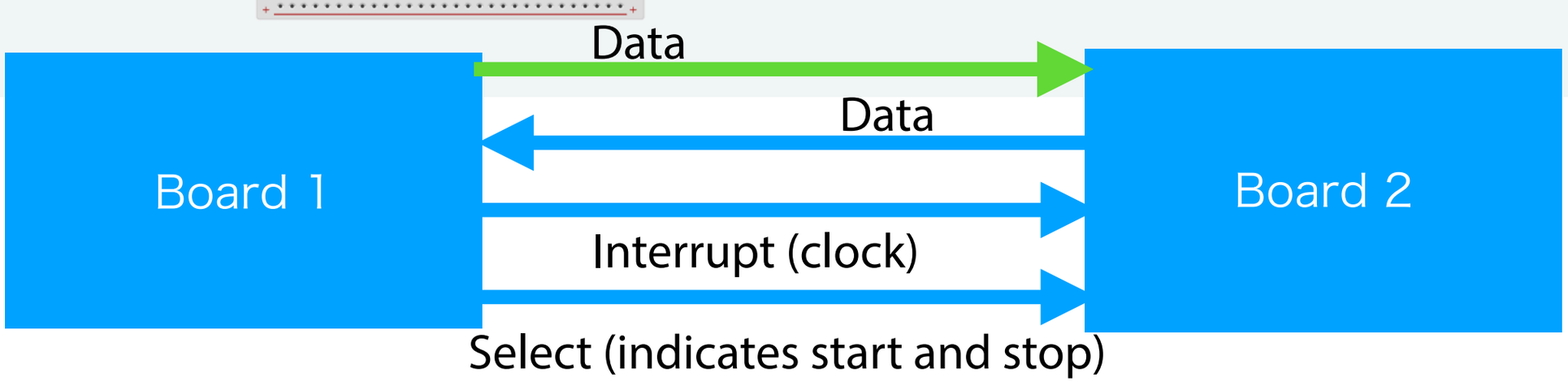
  (4)Hardware and software structure (how did you design it)

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

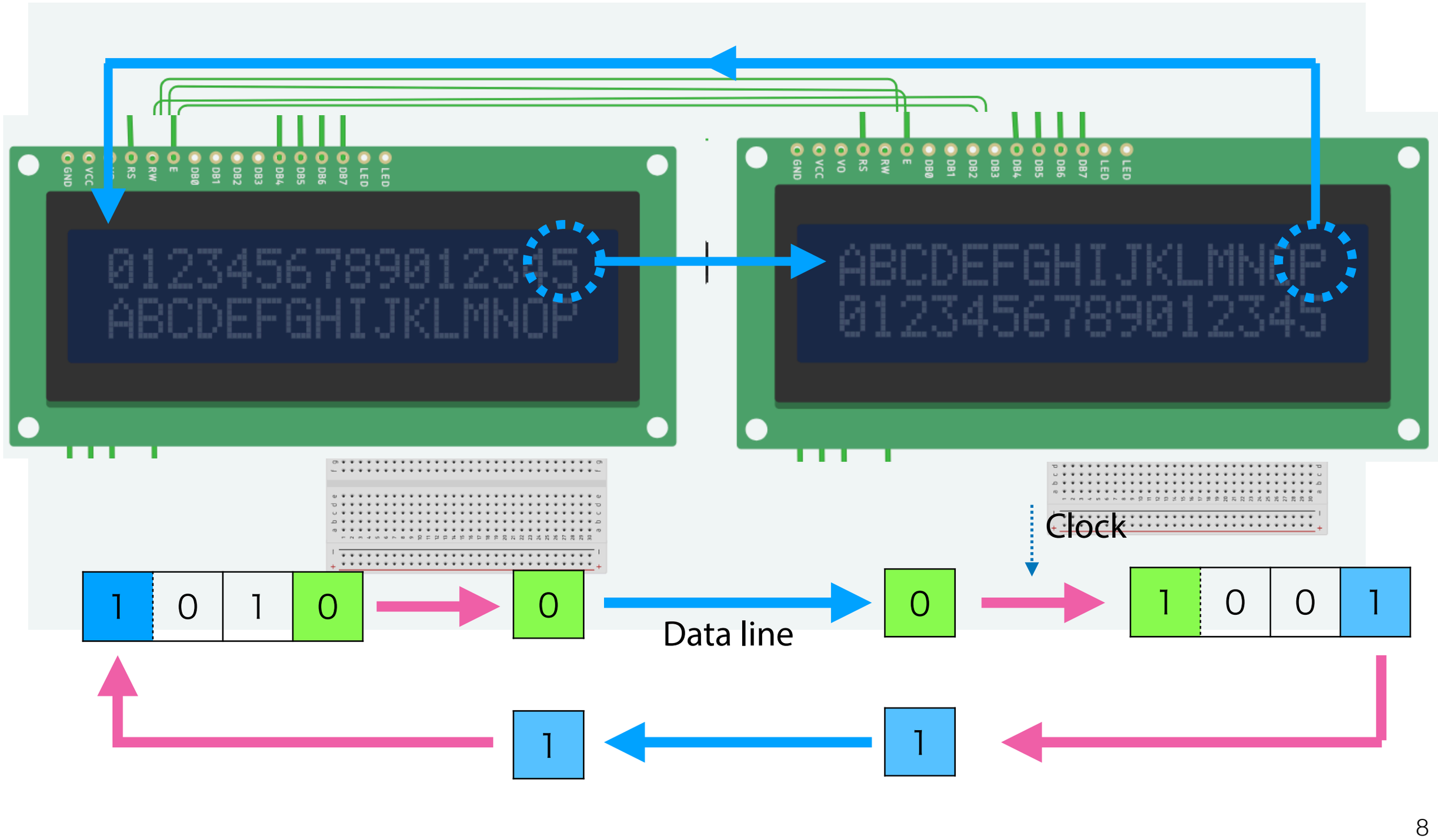
- **In (2), explain what the motivation for your design was.**
- **The (4) must include what you planned out and all your program listings with your comments.**
- **If you could not complete what you expected, the report should describe what you did.**
- **The report must be in English or Japanese.**

c) 4-Wire Serial Communication

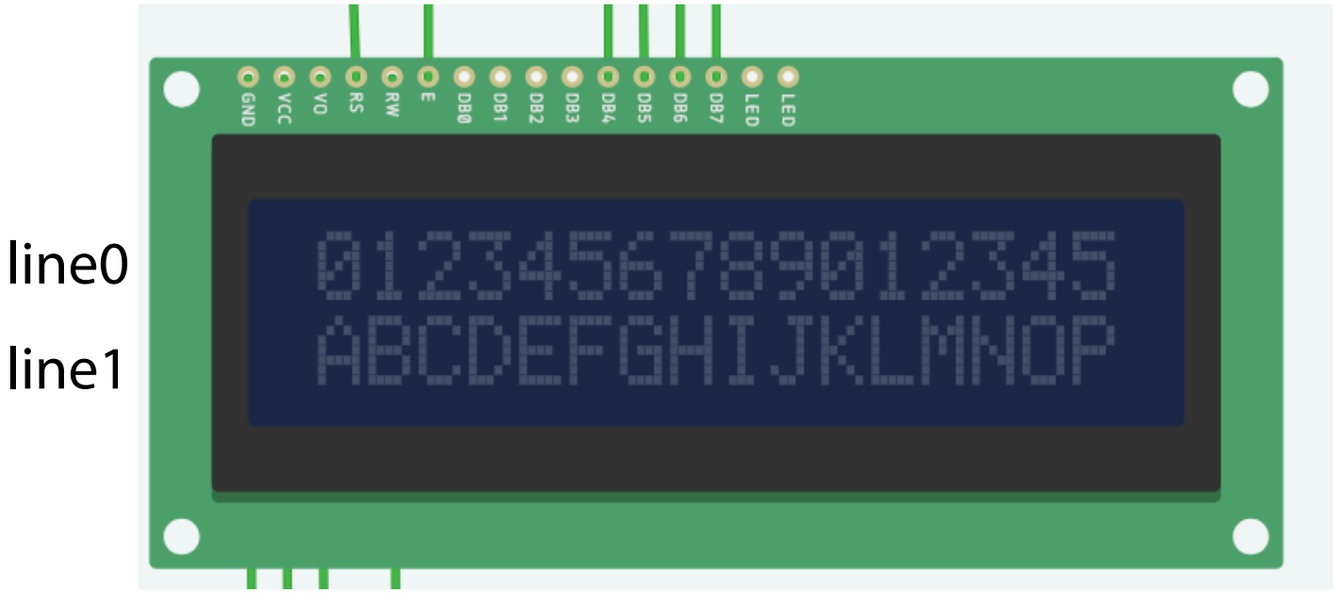


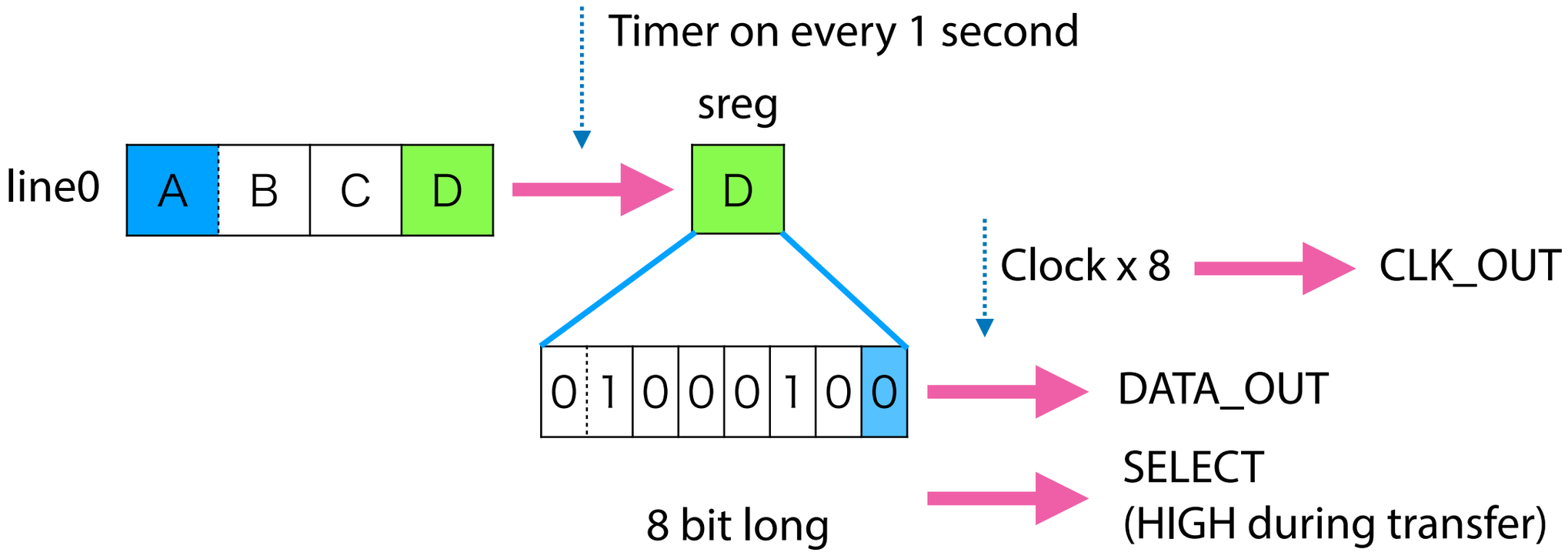


c) 4-Wire Serial Communication

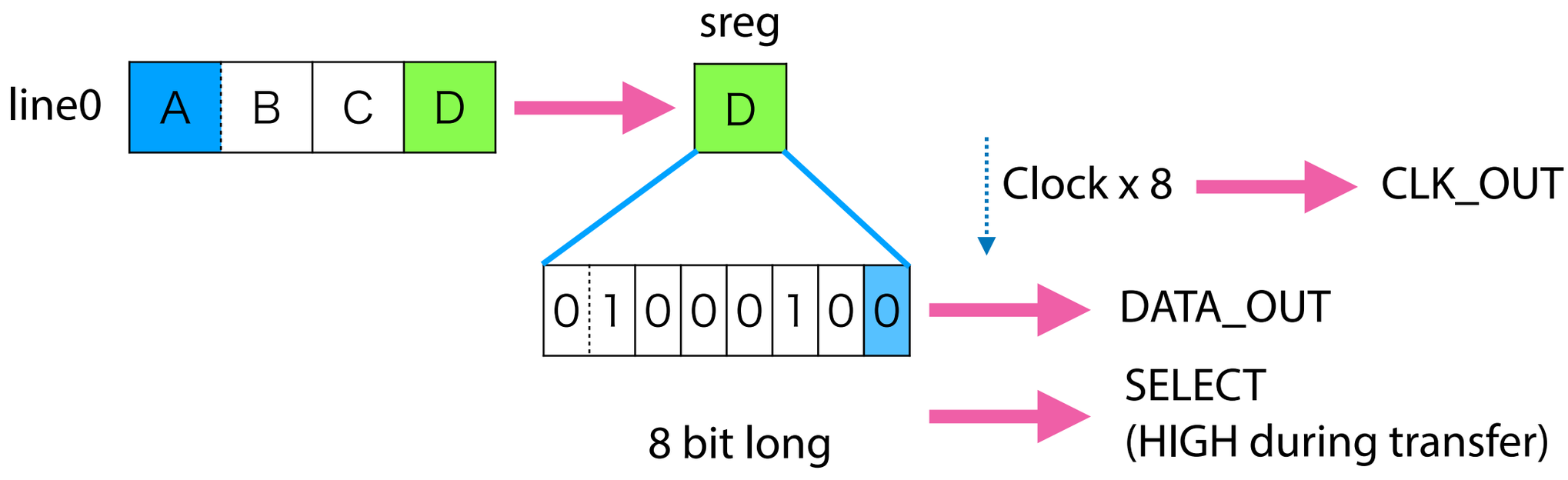


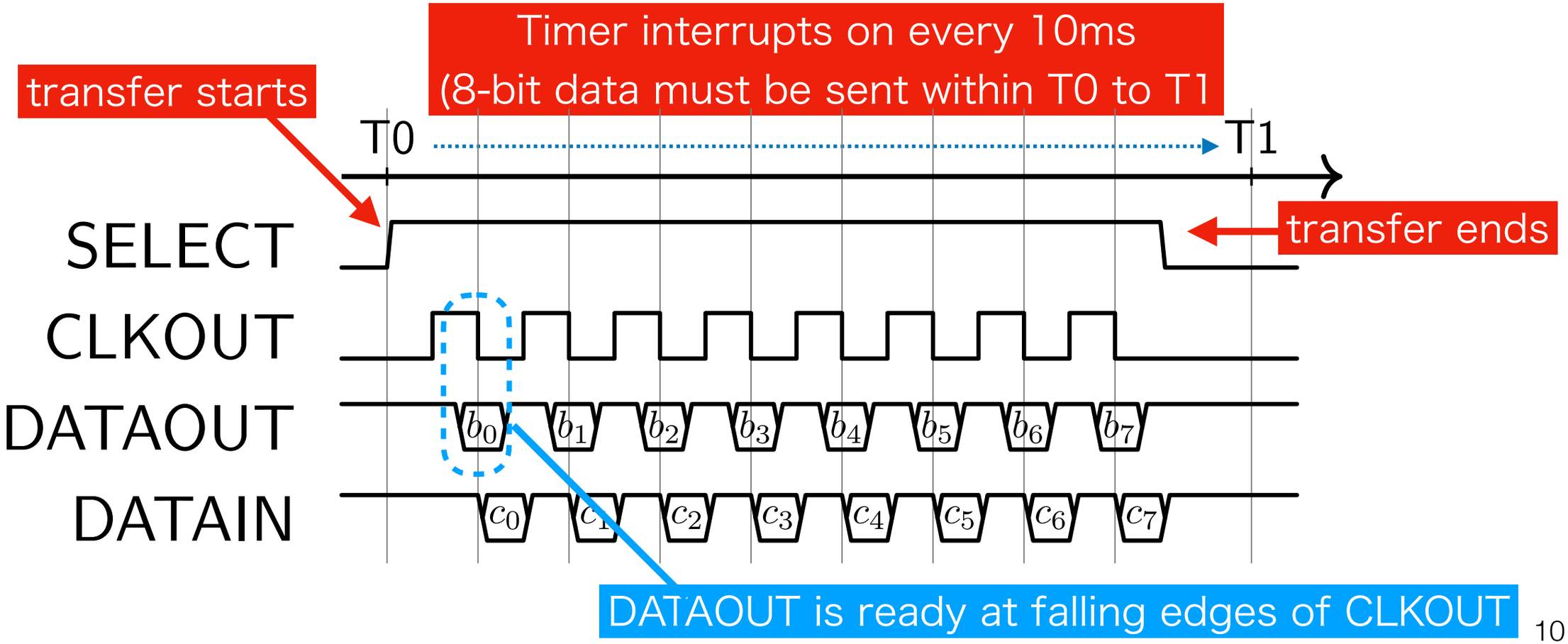
Design of Board 1



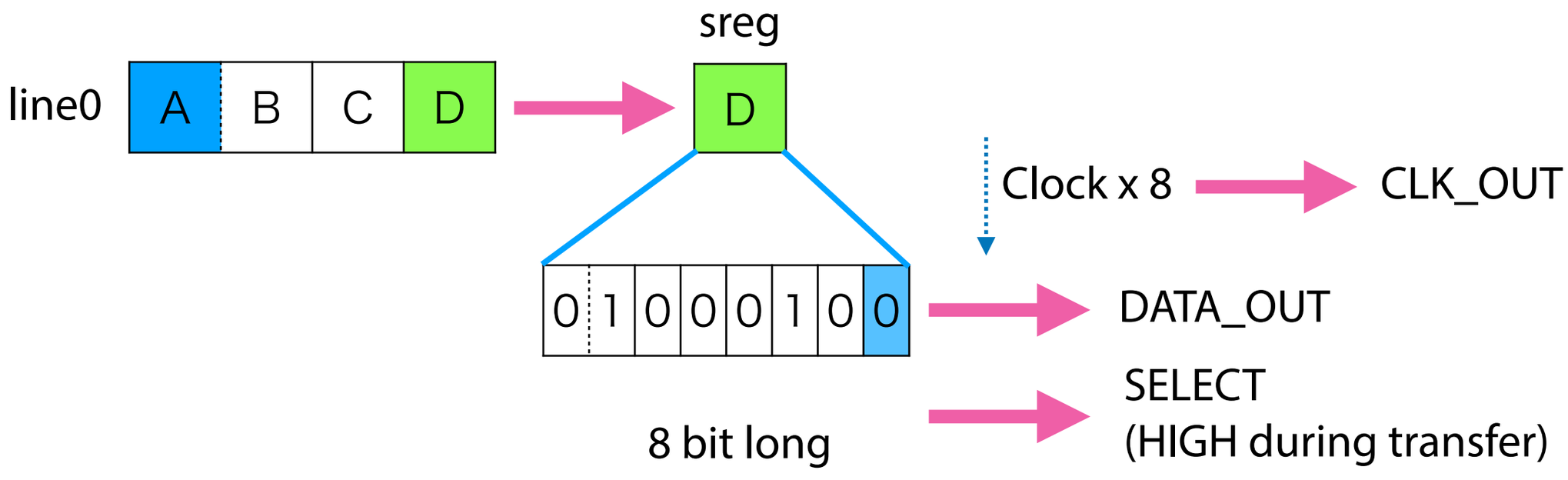


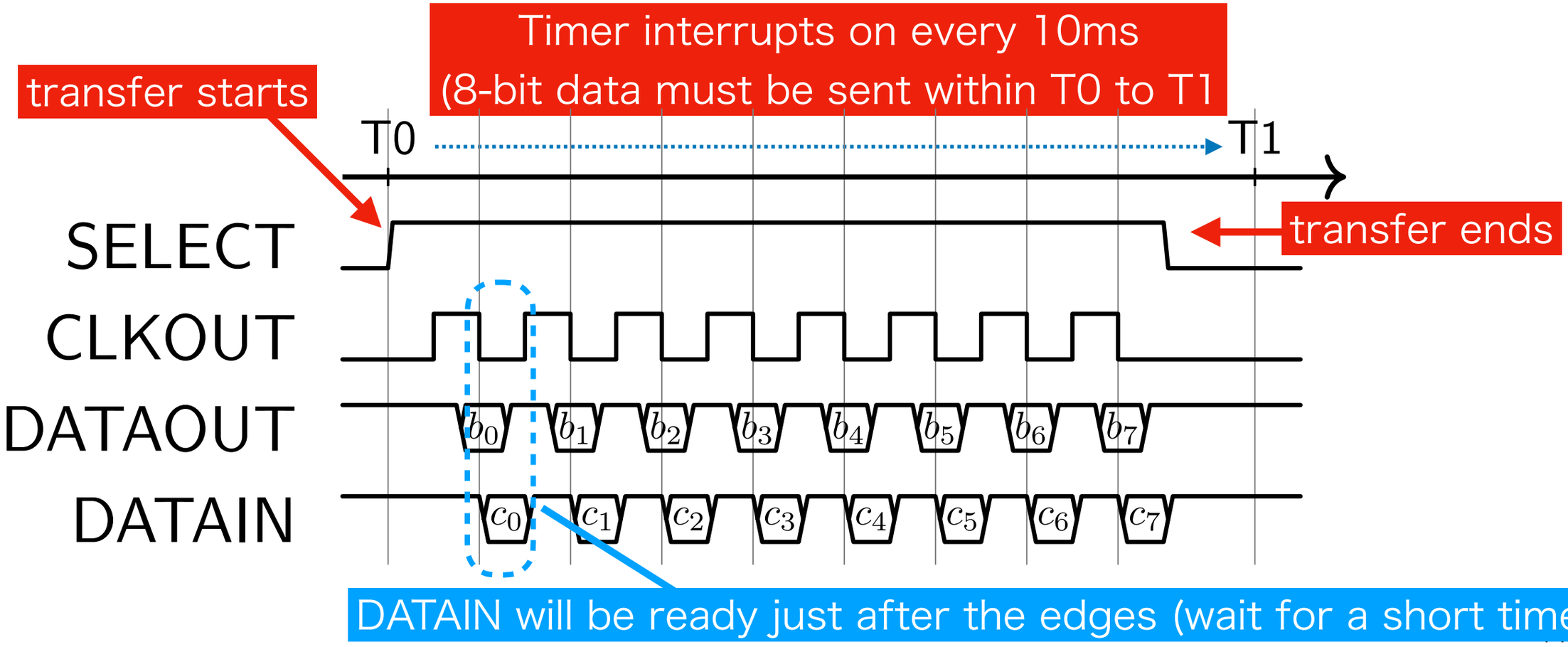
Timing Diagram on Board 1





Timing Diagram on Board 1





Board 1

```
/* shift register, 8 bit long */
unsigned char sreg;
char line0[] = "0123456789012345";
char line1[] = "ABCDEFGHIJKLMNOP";
char linebuf[]="                ";
int count;
int ready;

ISR (TIMER1_COMPA_vect) {
  int i;

  /* Check ready == 1 */
  if (!ready)
    return;
  /* send/recv in every 100 interrupts */
  if ((count++ % 100) != 0)
    return;

  digitalWrite(SELECT, HIGH);

  /* Copy line0 from the 2nd char except
     for the right-most. */
  memcpy(linebuf + 1, line0,
    sizeof(linebuf) - 2);
  /* Get the right-most. */
  sreg = line0[sizeof(line0) - 2];

  for (i = 0; i < BITLEN(sreg); i++) {
    digitalWrite(CLK_OUT, HIGH);

    /* Set LSB as data to be sent. */
    digitalWrite(DATA_OUT, sreg & 1);

    /* Generate a falling edge on CLK. */
    digitalWrite(CLK_OUT, LOW);

    /* Wait for a short time so that
       board 2 can update DATA_IN wire. */
    digitalWrite(CLK_OUT, LOW);
    digitalWrite(CLK_OUT, LOW);

    /* Read DATA_IN and update sreg */
    sreg >>= 1;
    if (digitalRead(DATA_IN) == HIGH)
      sreg |= 1 << (BITLEN(sreg) - 1);
  }
  digitalWrite(SELECT, LOW);

  /* Update the left-most char. */
  linebuf[0] = sreg;
  /* Update line0 */
  memcpy(line0, linebuf, sizeof(line0));
  update_lcd();
}
```

each bit in sreg → DATA_OUT

CLK → LOW

Timer interrupt handler on every 10 ms

Transfer on every 1 s

SELECT→HIGH

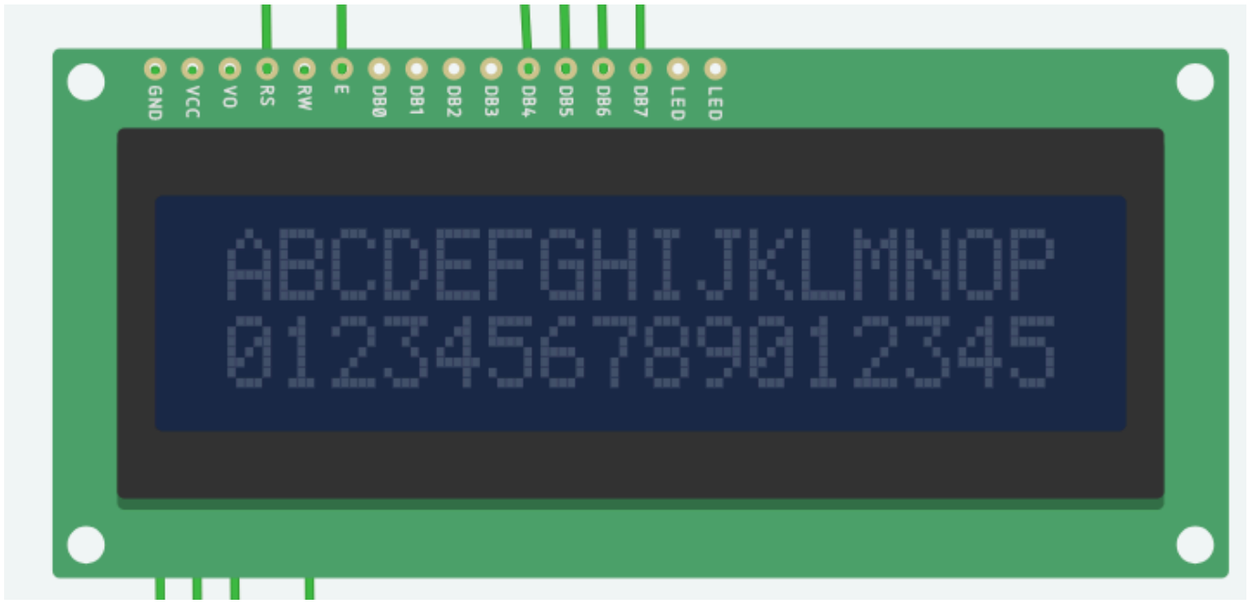
SELECT → LOW

sreg ← right-most char

Update chars on LCD

CLK → HIGH

Design of Board 2



line0

line1

LOW

SELECT
(HIGH during transfer)

sreg

CLK_IN

CLK_IN

CLK_IN

A

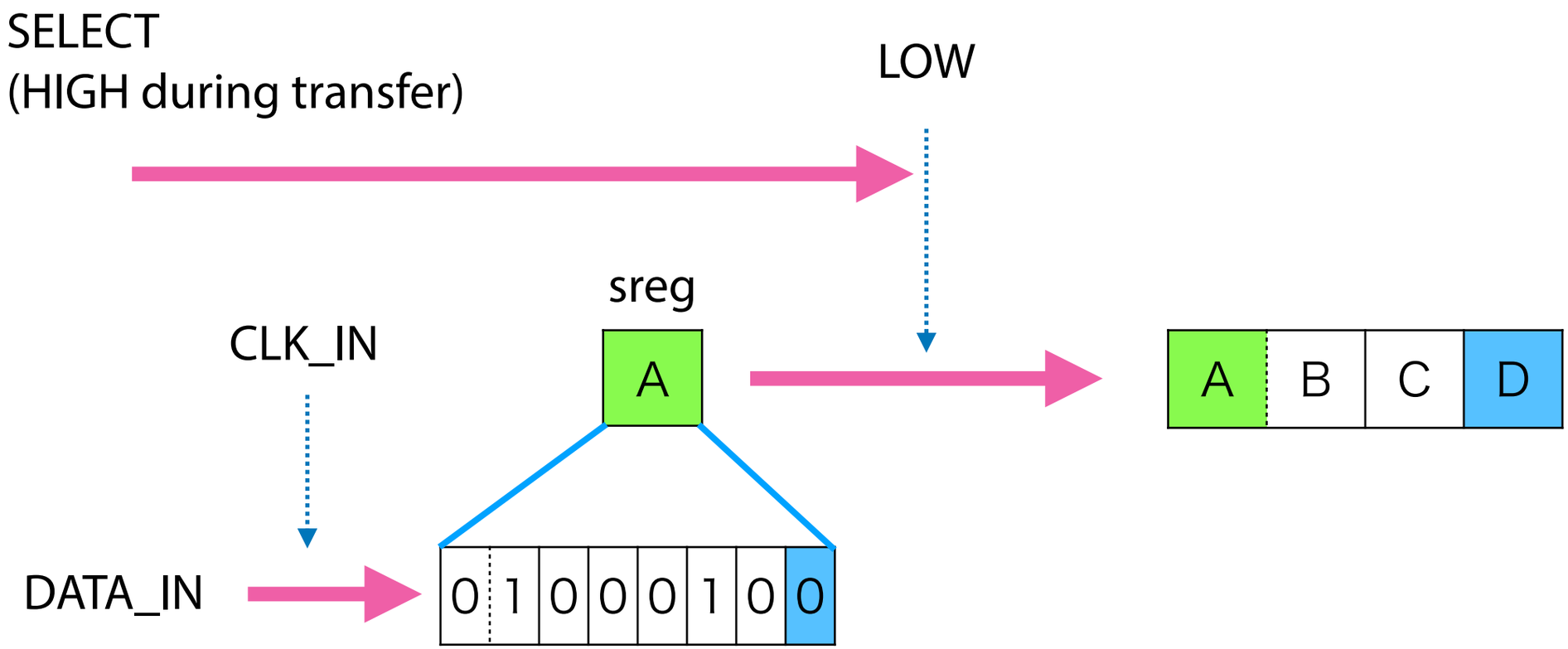
A | B | C | D

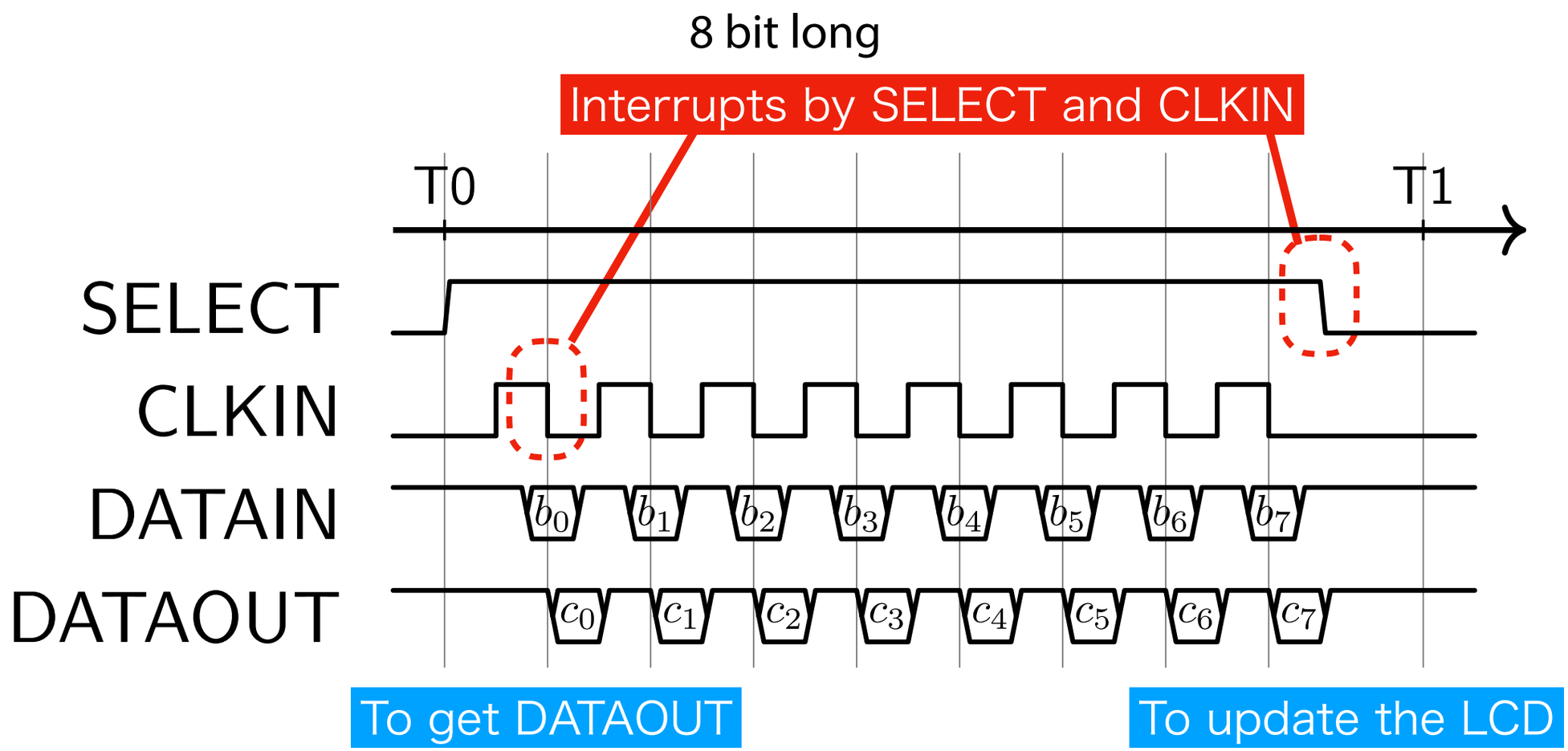
line0

DATA_IN

0 1 0 0 0 1 0 0

8 bit long


SELECT
(HIGH during transfer)
LOW
sreg
CLK_IN
A
A
B
C
D
DATA_IN
0 1 0 0 0 1 0 0


8 bit long
Interrupts by SELECT and CLKIN
T0
T1
SELECT
CLKIN
DATAIN
DATAOUT
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
c_0 c_1 c_2 c_3 c_4 c_5 c_6 c_7
To get DATAOUT
To update the LCD

Board 2

Characters for LCD

```
/* shift register, 8 bit long */
unsigned char sreg;
char line0[] = "ABCDEFGHIJKLMNOP";
char line1[] = "0123456789012345";
char linebuf[]="                ";
int receiving;
```

Interrupt handler for CLK_IN

```
void on_clk(void)
{
  /* Check if data is coming. */
  if (digitalRead(SELECT) != HIGH)
    return;
  /* Check if already receiving. */
  if (receiving == 0) {
    /* If not, initialize sreg. */
    sreg = line0[sizeof(line0) - 2];
 }

  /* Start send/recv */
  receiving++;
  /* Send LSB */
  digitalWrite(DATA_OUT, sreg & 1);
  /* Receive MSB */
  sreg >>= 1;
  sreg |= digitalRead(DATA_IN)
    << (BITLEN(sreg) - 1);
}
```

Interrupt handler for SELECT

```
void on_select(void)
{
  /* Finish send/recv */
  receiving = 0;

  /* Copy line0 from the 2nd char except
     for the right-most. */
  memcpy(linebuf + 1, line0,
    sizeof(linebuf) - 2);

  /* Update the left-mode */
  linebuf[0] = sreg;

  /* Update line0 */
  memcpy(line0, linebuf, sizeof(line0));
  update_lcd();
}
```

Board 2

```
/* shift register, 8 bit long */
unsigned char sreg;
char line0[] = "ABCDEFGHIJKLMNOP";
char line1[] = "0123456789012345";
char linebuf[]="                ";
int receiving;

void on_clk(void)
{
  /* Check if data is coming. */
  if (digitalRead(SELECT) != HIGH)
    return;

  /* Check if already receiving. */
  if (receiving == 0) {
    /* If not, initialize sreg. */
    sreg = line0[sizeof(line0) - 2];
  }

  /* Start send/recv */
  receiving++;
  /* Send LSB */
  digitalWrite(DATA_OUT, sreg & 1);
  /* Receive MSB */
  sreg >>= 1;
  sreg |= digitalRead(DATA_IN)
    << (BITLEN(sreg) - 1);
}
```

Characters for LCD

Interrupt handler for CLK_IN

"receiving == 0" means SELECT was high upon CLK interrupt.

LSB of sreg → DATA_OUT

MSB of sreg ← DATA_IN

```
void on_select(void)
{
  /* Finish send/recv */
  receiving = 0;

  /* Copy line0 from the 2nd char except
     for the right-most. */
  memcpy(linebuf + 1, line0,
    sizeof(linebuf) - 2);

  /* Update the left-mode */
  linebuf[0] = sreg;

  /* Update line0 */
  memcpy(line0, linebuf, sizeof(line0));
  update_lcd();
```

Board 2

```
/* shift register, 8 bit long */
unsigned char sreg;
char line0[] = "ABCDEFGHIJKLMNOP";
char line1[] = "0123456789012345";
char linebuf[]="                ";
int receiving;

void on_clk(void)
{
  /* Check if data is coming. */
  if (digitalRead(SELECT) != HIGH)
    return;

  /* Check if already receiving. */
  if (receiving == 0) {
    /* If not, initialize sreg. */
    sreg = line0[sizeof(line0) - 2];
 }

  /* Start send/recv */
  receiving++;
  /* Send LSB */
  digitalWrite(DATA_OUT, sreg & 1);
  /* Receive MSB */
  sreg >>= 1;
  sreg |= digitalRead(DATA_IN)
    << (BITLEN(sreg) - 1);
}
```

Characters for LCD

Interrupt handler for CLK_IN

Interrupt handler for SELECT

```
void on_select(void)
{
  /* Finish send/recv */
  receiving = 0;

  /* Copy line0 from the 2nd char except
     for the right-most. */
  memcpy(linebuf + 1, line0,
    sizeof(linebuf) - 2);

  /* Update the left-mode */
  linebuf[0] = sreg;

  /* Update line0 */
  memcpy(line0, linebuf, sizeof(line0));
  update_lcd();
}
```

SELECT becomes LOW

Update LCD

Check

- **Interrupts**
  - Interrupts triggered by the timer device and how to configure them.
    - Do not use delay() in a timer interrupt handler.
  - Interrupts triggered by GPIO pins. **Note that 2 and 3 pins only**.
- **How to design the serial communication protocol**
  - Be aware of wires and voltage change timings.
  - Precise timing (real-time-ness) is important for embedded systems.

State Machine

State Machine

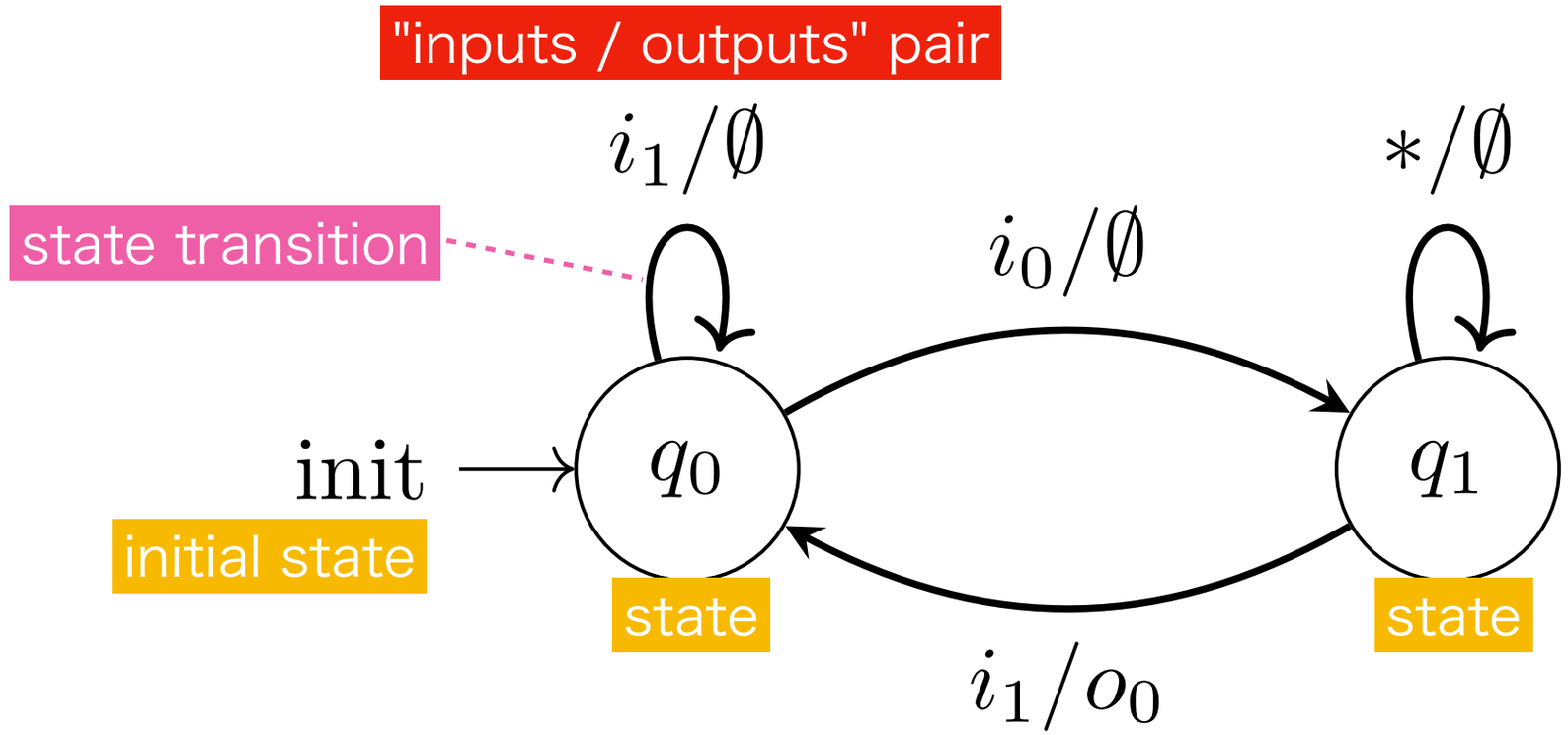
- An event-driven system can be modeled by using FSM (finite state machine or finite automaton).
- The actors are "inputs", "states", "outputs"

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

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

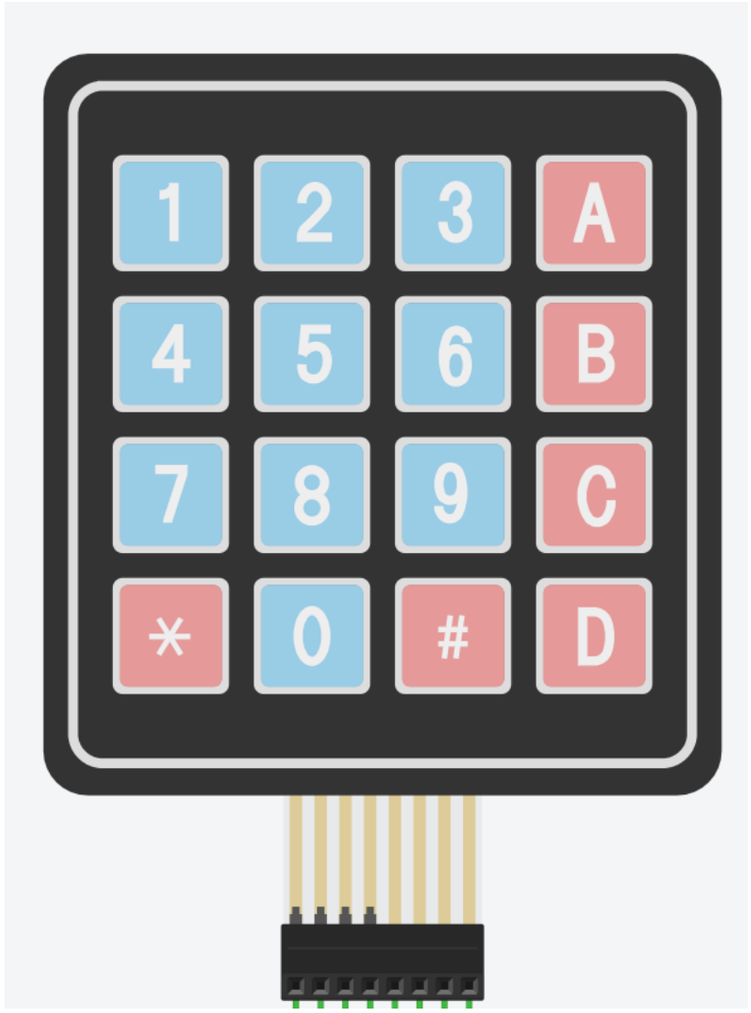
State Transition Diagram



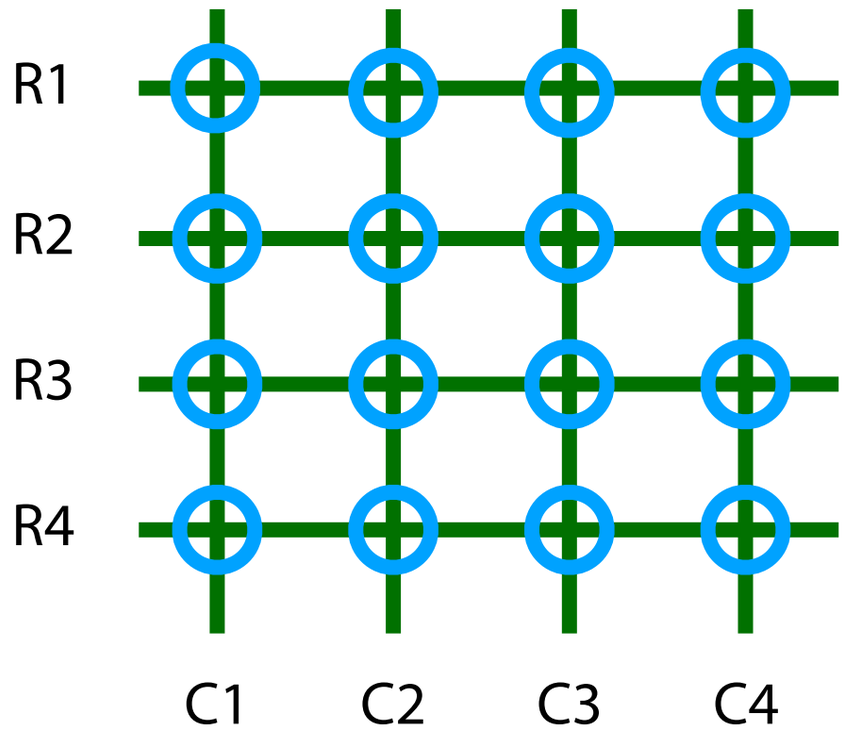
$$\{\{q_0, q_1\}, \{i_0, i_1\}, \{o_0\}, T, G, s\}$$

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

Example: Matrix Switch (keypad)



8 pins



8 pins

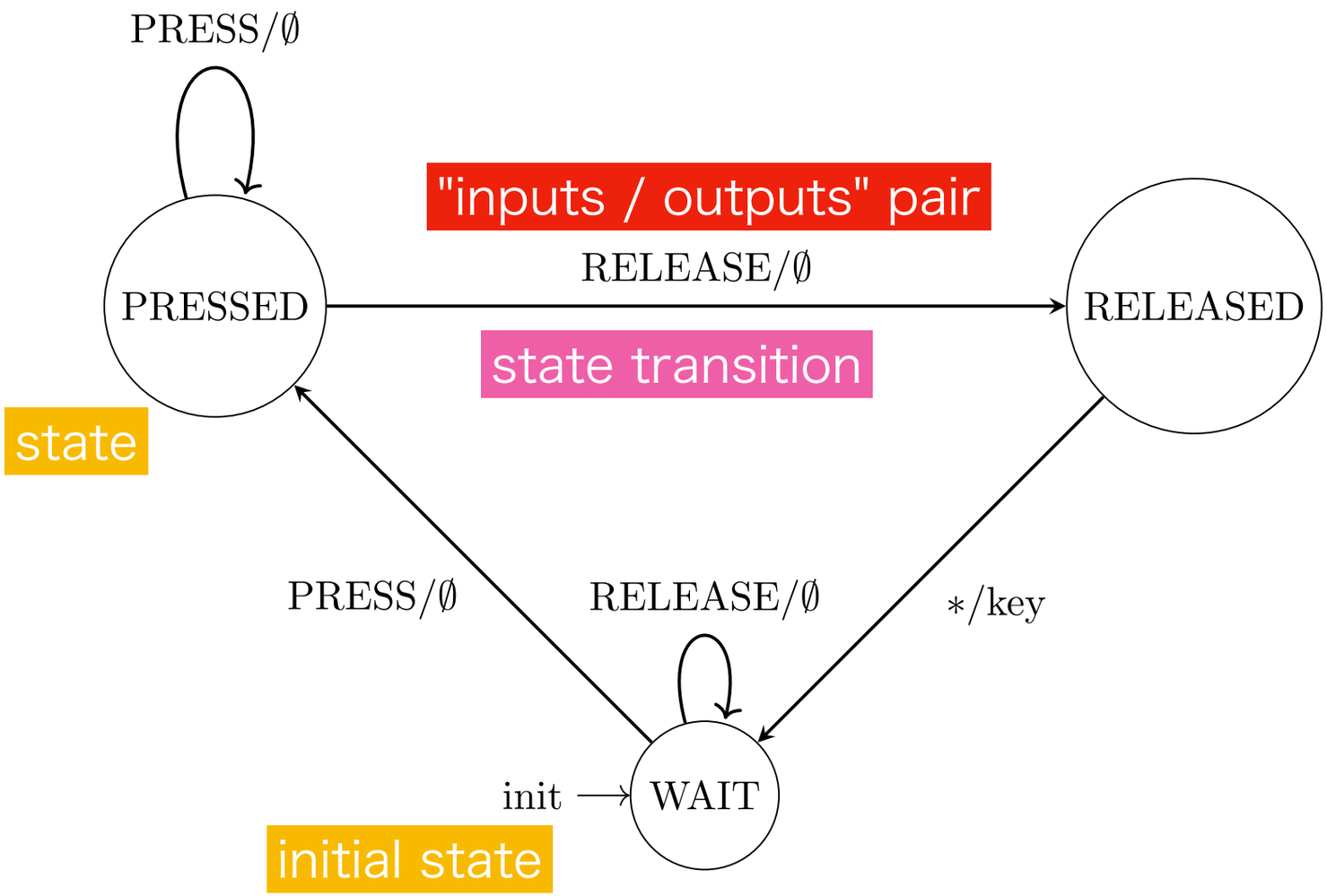
- The complexity of handling a keypad
- Needs to check 8 wires every time.

```
/* 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 */
}
```

Example: Matrix Switch (keypad)

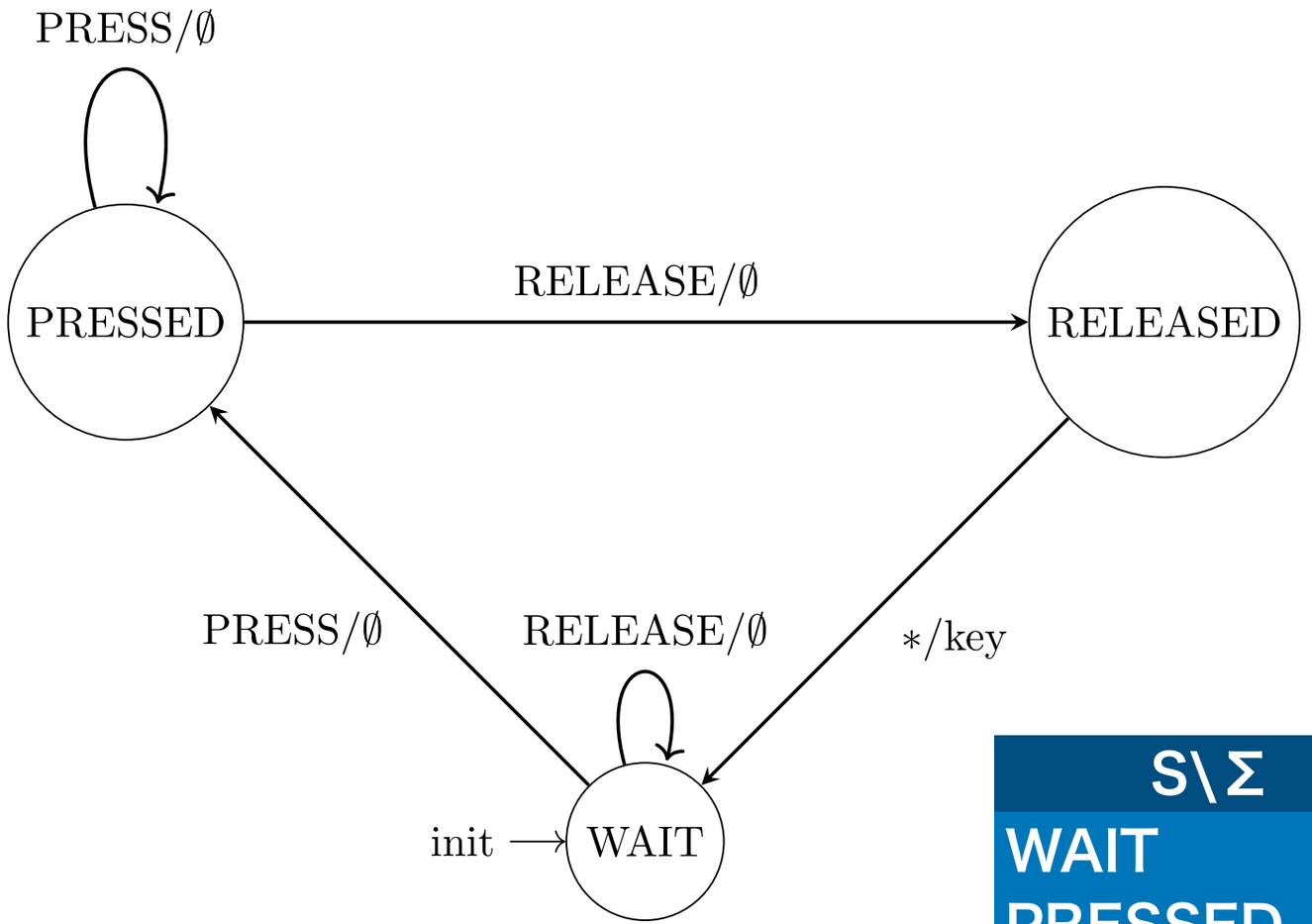
- Complexity of handling a keypad (cont'd)
  - Two events of pressing and releasing must be detected
    - Checking only a press is not enough
    - Both checking pressing or releasing requires scanning of the 8 wires

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

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

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

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

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

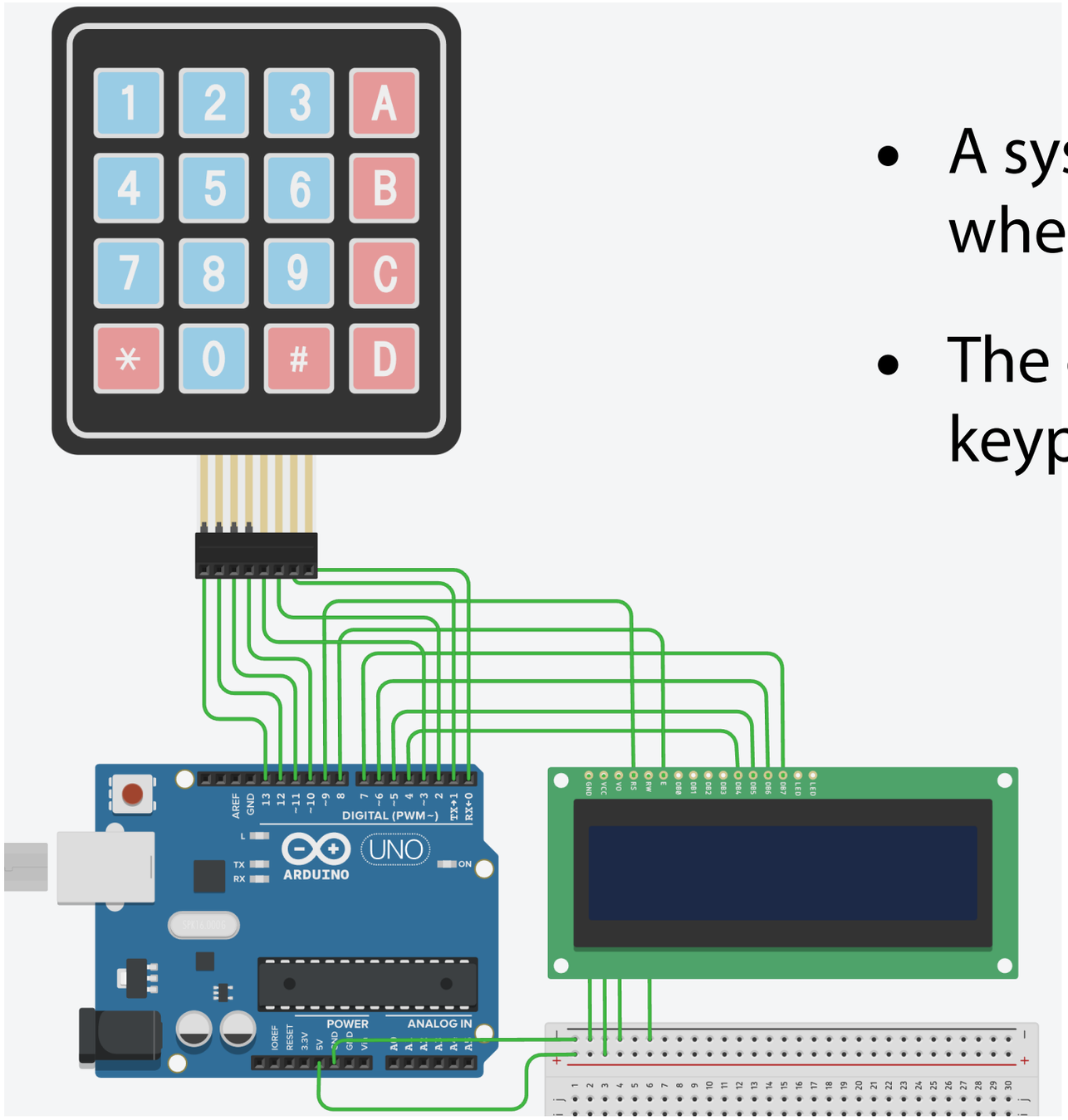
- A state transition table can be directly implemented as an array.
- This guarantees that every possible transitions are covered on the system.

```
void
output(int s, int i, char k)
{
  if (s == S_RELEASED)
    update_lcd1(k);
}

int
get_input(void)
{
  char k;

  k = scan_keypad();
  if (k == 0)
    return I_RELEASE;
  else {
    k0 = k;
    return I_PRESS;
  }
}

/* Current state */
enum state_t state = S_WAIT;

void
loop() {
  enum input_t i;

  i = (enum input_t)get_input();
  state = s_next[state][i];
  output(state, i, k0);
  delay(50);
}
```

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.

A Complete Example



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

Demo

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

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

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

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

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

```
/* 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 */
}
```

```
/* 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
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);
}
```

```
/* NOTE: the argument type should be enum
state_t and intput_t */
void
output(int s, int i, char k)
{

  if (s == S_RELEASED)
    update_lcd1(k);
}

/* NOTE: the return type should be enum
input_t */
int
get_input(void)
{
  char k;

  k = scan_keypad();
  if (k == 0)
    return I_RELEASE;
  else {
    k0 = k;
    return I_PRESS;
  }
}
```

```
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 */
enum state_t state = S_WAIT;
```

```
void
loop() {
  enum input_t i;

  i = (enum input_t)get_input();
  state = s_next[state][i];
  output(state, i, k0);
  delay(50);
}
```

More compact implementation

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

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

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

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

The contents of the table are in s_next and output()

```
void
output(int s, int i, char k)
{

  if (s == S_RELEASED)
    update_lcd1(k);
}
```

More compact implementation

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

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

enum 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

```
void
output(int s, int i, char k)
{

  if (s == S_RELEASED)
    update_lcd1(k);
}
```

```
#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 */
}
```

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

  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);
}
```

Caveats about the Simulator

- **enum tag does not work for function declarations**

correct but does not work

```
enum input_t {
  I_RELEASE,
  I_PRESS
};

enum input_t
get_input(void)
{
   return I_RELEASE;
}

void
loop(void) {
  enum input_t i;

  i = get_input();
}
```

```
enum input_t {
  I_RELEASE,
  I_PRESS
};

int
get_input(void)
{
   return I_RELEASE;
}

void
loop(void) {
  enum input_t i;

  i = (enum input_t)get_input();
}
```

works

Caveats about GPIO

- **GPIO pin0 and pin1**
  - Serial.begin() uses these two pins.
- **GPIO pin2 and pin3**
  - You can apply attachInterrupt() only to these two pins.
- **pinMode()**
  - You need to call pinMode() for all of the pins that you want to use. If you forget it, you will get a malformed behavior.

Time for Your Project

- Read the example answers and try them on the simulator.
  - If you finish them, try implementing a keypad+LCD+blinker, which has a blinking "＊" on the first line.
- Start to plan your end-term project.
  - Complete all of the past exercises yourself, even if not for evaluation. Experience in development is important.
- Make backups of your designs on your local machine.
- Keep connected to your Zoom session. An ending announcement will be made just before 16:50.
- Do not create a new account for yourself on TinkerCAD. Use your nickname account (ask the teaching assistants for your nickname if you forget it). **Non-nickname accounts will be removed without notice.**