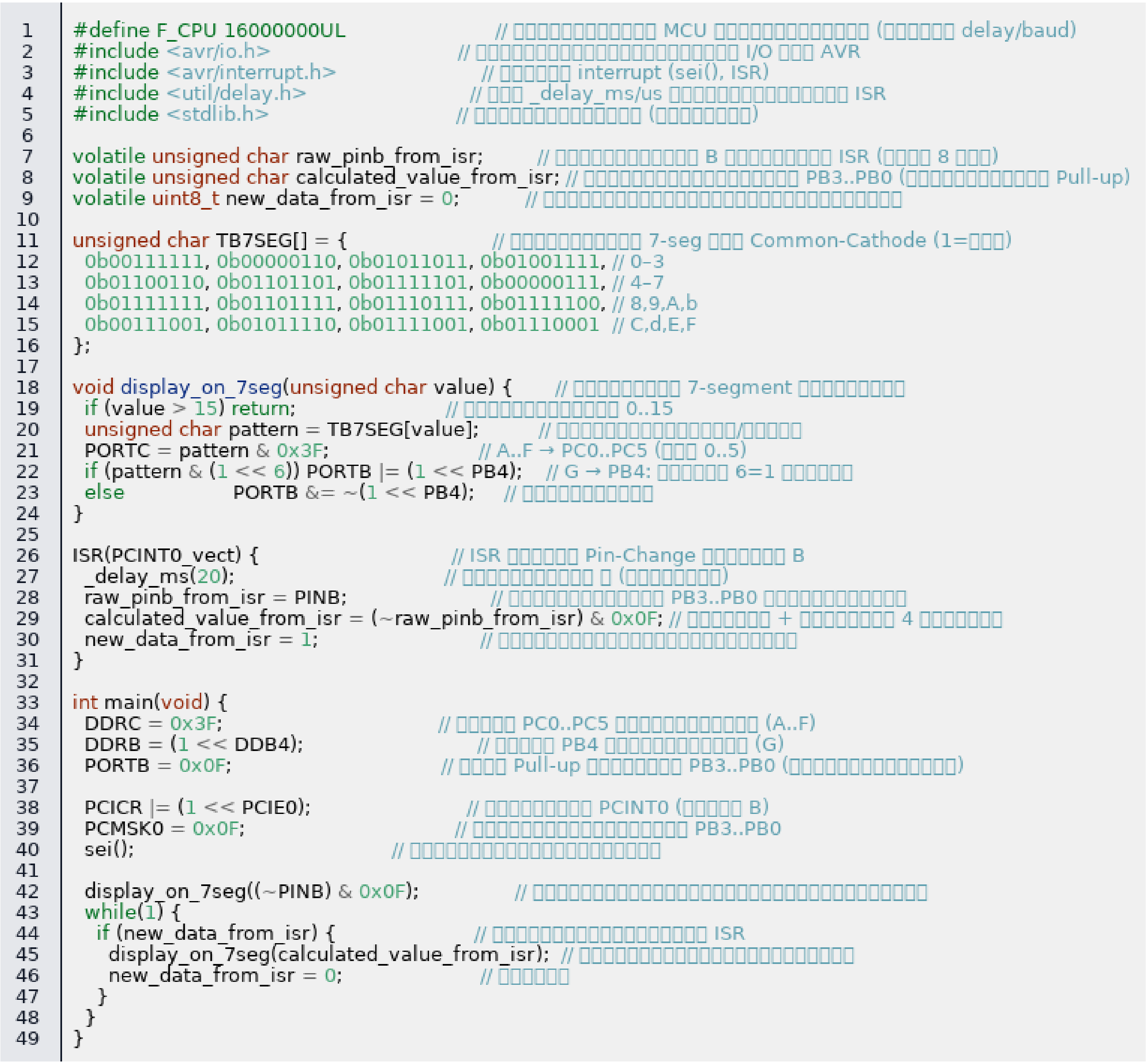
Checkpoint 3.1 — AVR PCINT0 + 7■Segment (Light Theme)

Light theme • Thai-capable font verified • USART removed



```c
#define F_CPU 16000000UL                    // □□□□□□□□□□□□□□ MCU □□□□□□□□□□□□□□□ (□□□□□□□ delay/baud)
#include <avr/io.h>                      // □□□□□□□□□□□□□□□□□□□□□□□□□□□ I/O □□□ AVR
#include <avr/interrupt.h>                  // □□□□□□□ interrupt (sei(), ISR)
#include <util/delay.h>                   // □□□ _delay_ms/us □□□□□□□□□□□□□□□□□□ ISR
#include <stdlib.h>                      // □□□□□□□□□□□□□□□□□ (□□□□□□□□□□)

volatile unsigned char raw_pinb_from_isr;         // □□□□□□□□□□□□□□ B □□□□□□□□□□□ ISR (□□□□□ 8 □□□)
volatile unsigned char calculated_value_from_isr; // □□□□□□□□□□□□□□□□□□□□□□ PB3..PB0 (□□□□□□□□□□□□□□□ Pull-up)
volatile uint8_t new_data_from_isr = 0;          // □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

unsigned char TB7SEG[] = {                  // □□□□□□□□□□□□□ 7-seg □□□ Common-Cathode (1=□□□)
  0b00111111, 0b00000110, 0b01011011, 0b01001111, // 0–3
  0b01100110, 0b01101101, 0b01111101, 0b00000111, // 4–7
  0b01111111, 0b01101111, 0b01110111, 0b01111100, // 8,9,A,b
  0b00111001, 0b01011110, 0b01111001, 0b01110001  // C,d,E,F
};

void display_on_7seg(unsigned char value) {      // □□□□□□□□□□□ 7-segment □□□□□□□□□□□
  if (value > 15) return;                      // □□□□□□□□□□□□□□□□ 0..15
  unsigned char pattern = TB7SEG[value];          // □□□□□□□□□□□□□□□□□□/□□□□□□
  PORTC = pattern & 0x3F;                     // A..F → PC0..PC5 (□□□ 0..5)
  if (pattern & (1 << 6)) PORTB |= (1 << PB4);     // G → PB4: □□□□□□□ 6=1 □□□□□□□
  else                 PORTB &= ~(1 << PB4);     // □□□□□□□□□□□□□
}

ISR(PCINT0_vect) {                          // ISR □□□□□□□ Pin-Change □□□□□□□□ B
  _delay_ms(20);                             // □□□□□□□□□□□□□□ □ (□□□□□□□□□□)
  raw_pinb_from_isr = PINB;                       // □□□□□□□□□□□□□□□□ PB3..PB0 □□□□□□□□□□□□□□□
  calculated_value_from_isr = (~raw_pinb_from_isr) & 0x0F; // □□□□□□□□□ + □□□□□□□□□□ 4 □□□□□□□□□
  new_data_from_isr = 1;                        // □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
}

int main(void) {
  DDRC = 0x3F;                               // □□□□□□ PC0..PC5 □□□□□□□□□□□□□□□ (A..F)
  DDRB = (1 << DDB4);                           // □□□□□□ PB4 □□□□□□□□□□□□□□□ (G)
  PORTB = 0x0F;                              // □□□□□ Pull-up □□□□□□□□□□ PB3..PB0 (□□□□□□□□□□□□□□□□□□)

  PCICR |= (1 << PCIE0);                        // □□□□□□□□□□□ PCINT0 (□□□□□□ B)
  PCMSK0 = 0x0F;                              // □□□□□□□□□□□□□□□□□□□□□□□ PB3..PB0
  sei();                                // □□□□□□□□□□□□□□□□□□□□□□□□□□□

  display_on_7seg((~PINB) & 0x0F);              // □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
  while(1) {
    if (new_data_from_isr) {                    // □□□□□□□□□□□□□□□□□□□□□□ ISR
      display_on_7seg(calculated_value_from_isr);  // □□□□□□□□□□□□□□□□□□□□□□□□□□□□
      new_data_from_isr = 0;                    // □□□□□□□
    }
  }
}
```

Checkpoint 3.2 — AVR INT0 Counter 0–9 + 7■Segment (Light Theme)

Light theme • Thai-capable font verified • USART removed

```
#define F_CPU 16000000UL                    // □□□□□□□□□□□□□□□□□□□□□□□□□
#include <avr/io.h>                      // □□□□□□□□□□□□□□□□□□□□□
#include <avr/interrupt.h>                  // □□□□ INT0 □□□□ sei()
#include <avr/pgmspace.h>                     // □□□□□□□□□□□□□□□□□□□□□□□□□□□ pgm_read_byte
#include <util/delay.h>                       // □□□□□□□□□□□□□□□□□□□□□□□ ISR

volatile unsigned char count = 0;               // □□□□□□□□□□□□ 0..9 □□□□□□□□□□□□
volatile uint8_t new_count_to_display = 0;        // □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

const unsigned char seven_seg_table[16] PROGMEM = { // □□□□□□□ 7-seg □□□□□□□□□□□□□
  0b00111111, 0b00000110, 0b01011011, 0b01001111, // 0–3
  0b01100110, 0b01101101, 0b01111101, 0b00000111, // 4–7
  0b01111111, 0b01101111, 0b01110111, 0b01111100, // 8,9,A,b
  0b00111001, 0b01011110, 0b01111001, 0b01110001  // C,d,E,F
};

void display_count(unsigned char number) {        // □□□□□□□□□□□□□□ 0..15
  if (number > 15) return;                       // □□□□ index □□□□□
  unsigned char pattern = pgm_read_byte(&seven_seg_table[number]); // □□□□□□□□□□□□□□□□□□
  PORTC = pattern & 0x3F;                          // A..F → PC0..PC5
  if (pattern & (1 << 6)) PORTB |= (1 << PB0);    // G → PB0
  else                  PORTB &= ~(1 << PB0);
}

ISR(INT0_vect) {                                // □□□□□□□□□□□□□□□□□□□□□□ PD2 (INT0)
  _delay_ms(10);                                // □□□□□□□□□□□□□□□ □
  if (!(PIND & (1 << PIND2))) {                    // □□□□□□□□□□□□□□□□□□□□□□□□□□□ (active-LOW)
    count++; if (count >= 10) count = 0;           // □□□□□□□ 0..9
    new_count_to_display = 1;                      // □□□□□□□□□□□□□□□□□□□□□□□□□□□□□
  }
}

int main(void) {
  DDRC = 0x3F;                                 // PC0..PC5 = □□□□□□□□□□□
  DDRB = (1 << DDB0);                              // PB0 = □□□□□□□□□□□ (G)
  DDRD &= ~(1 << PD2);                              // PD2 = □□□□□□□□ (□□□□□ INT0)
  PORTD |= (1 << PD2);                           // □□□□□ Pull-up □□□□ PD2

  EICRA = (1 << ISC01);                          // INT0 □□□□□□□□□□□□ (FALLING)
  EIMSK = (1 << INT0);                           // □□□□□□□□ INT0
  sei();                                 // □□□□□□□□□□□□□□□□□

  display_count(0);                             // □□□□□□□□□□□□□□□□ 0
  while (1) {
    if (new_count_to_display) {                   // □□□□□□□□□□□□□□□□ ISR
      new_count_to_display = 0;                   // □□□□□□□□□□□□□
      display_count(count);                    // □□□□□□□□□□□
    }
  }
}
```

Checkpoint 3.3 — Arduino Keypad 4×4 + 7■Segment (Light Theme)

Light theme • Thai-capable font verified • USART removed

```
// Arduino UNO: Keypad 4x4 + 7-Segment (1 □□□□) — □□□□□ Serial □□□□□□
const int segmentA = 14; const int segmentB = 15;  // □□□ A0..A1 → A, B
const int segmentC = 16; const int segmentD = 17;  // □□□ A2..A3 → C, D
const int segmentE = 18; const int segmentF = 19;  // □□□ A4..A5 → E, F
const int segmentG = 12; const int seg1_enable = 13; // G □□□□□ enable □□□□

const int rowPins[4] = {4, 5, 6, 7};              // □□□□□□□□□□□□: □□□□□□□□ OUTPUT □□□□□□□□ LOW
const int colPins[4] = {8, 9, 10, 11};             // □□□□□□□□□□□□□□□□: □□□ INPUT_PULLUP
const int interruptPin = 3;                       // INT1 □□□□□□□□□□□□□□□□□□□□□□□□

volatile bool keypress_detected = false;           // □□□□□ ISR □□□□□□□□□□□□□□□
char keys[4][4] = {                               // □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
  {'D','C','B','A'}, {'F','9','6','3'}, {'0','8','5','2'}, {'E','7','4','1'}
};

void onKeyPress(){ keypress_detected = true; }      // ISR: □□□□□□□□□□ □ □□□□□□□□□□□□□□□□

char scanKeypad(){                                // □□□□□□□□□/□□□□□□□□□□□□□□□□□□
  char k = '\0';                                  // □□□□□ = '\0'
  for (int r=0;r<4;r++){                          // □□□□□□□□□□□
    for (int i=0;i<4;i++) digitalWrite(rowPins[i],HIGH); // □□□□□□□□□□□□
    digitalWrite(rowPins[r],LOW);                   // □□□□□□□□□□□□□□□□□ LOW
    for (int c=0;c<4;c++){                        // □□□□□□□□□□□□□□□
      if (digitalRead(colPins[c])==LOW){            // LOW = □□□□ r,c □□□□□
        delay(20); k = keys[r][c];                // □□□□□□□□ + □□□□□□□□
        while(digitalRead(colPins[c])==LOW);         // □□□□□□□□□□□□□□ (□□□ repeat)
        goto found;
      }
    }
  }
found:
  for (int i=0;i<4;i++) digitalWrite(rowPins[i],LOW); // □□□□□□□□□□□□ LOW (idle)
  return k;
}

void seg(bool a,bool b,bool c,bool d,bool e,bool f,bool g){ // □□□□□□□□□□□□ segment
  digitalWrite(segmentA,a); digitalWrite(segmentB,b);
  digitalWrite(segmentC,c); digitalWrite(segmentD,d);
  digitalWrite(segmentE,e); digitalWrite(segmentF,f);
  digitalWrite(segmentG,g);
}

void displayHex(char key){                         // □□□□□□□□□□□□□ 0..F □□□□□□ 7-seg
  switch(key){
    case '0': seg(1,1,1,1,1,1,0); break; case '1': seg(0,1,1,0,0,0,0); break;
    case '2': seg(1,1,0,1,1,0,1); break; case '3': seg(1,1,1,1,0,0,1); break;
    case '4': seg(0,1,1,0,0,1,1); break; case '5': seg(1,0,1,1,0,1,1); break;
    case '6': seg(1,0,1,1,1,1,1); break; case '7': seg(1,1,1,0,0,0,0); break;
    case '8': seg(1,1,1,1,1,1,1); break; case '9': seg(1,1,1,1,0,1,1); break;
    case 'A': seg(1,1,1,0,1,1,1); break; case 'B': seg(0,0,1,1,1,1,1); break;
    case 'C': seg(1,0,0,1,1,1,0); break; case 'D': seg(0,1,1,1,1,0,1); break;
    case 'E': seg(1,0,0,1,1,1,1); break; case 'F': seg(1,0,0,0,1,1,1); break;
    default: seg(0,0,0,0,0,0,0); break;
  }
}

void setup(){
  pinMode(segmentA,OUTPUT); pinMode(segmentB,OUTPUT); pinMode(segmentC,OUTPUT); // □□□□□□ 7-seg
  pinMode(segmentD,OUTPUT); pinMode(segmentE,OUTPUT); pinMode(segmentF,OUTPUT);
  pinMode(segmentG,OUTPUT); pinMode(seg1_enable,OUTPUT); digitalWrite(seg1_enable,HIGH); // □□□□□□□□

  for(int i=0;i<4;i++){ pinMode(rowPins[i],OUTPUT); digitalWrite(rowPins[i],LOW);} // □□□□□□□□□□□□□□□□
  for(int i=0;i<4;i++)  pinMode(colPins[i],INPUT_PULLUP);                // □□□□□□□□□□□□□□□□□□□□□□□□□□

  pinMode(interruptPin,INPUT_PULLUP);                           // □□□□□□□□□□ active-LOW
  attachInterrupt(digitalPinToInterrupt(interruptPin), onKeyPress, FALLING);     // □□□□□□□□□□□□□□□
}

void loop(){
  if(keypress_detected){ keypress_detected=false; char k=scanKeypad();          // □□□□□□□□□□□□□□□
    if(k!='\0') displayHex(k);                                 // □□□□□□□□□□□□□
  }
}
```