```
#include <avr/io.h>
#include <avr/interrupt.h>

#define LED_MASK ((1 << PB1) | (1 << PB2) | (1 << PB3)) // for pins 9,
10, and 11
//A bitmask or LEDmask is just a number you use to turn specific bits
on or off.

// Three modulation speeds (Timer1 prescaler = 1024)
const uint16_t OCR_SLOW = 7812;  // 2 Hz
const uint16_t OCR_MED  = 781;   // 20 Hz
const uint16_t OCR_FAST = 155;   // 100 Hz

volatile uint8_t mode = 0;       // 0 = all blink, 1 = chase, 2 =
binary
volatile uint8_t stepIndex = 0;  // step for chase/binary, step index
is used as a frame counter

// ---------------- Timer1 Compare A Interrupt ----------------
ISR(TIMER1_COMPA_vect) {    // Runs automatically every time Timer1
reaches OCR1A (OCR1A = Timer1 compare register)
  switch (mode) {
    case 0:  // Mode 0: ALL THREE BLINK TOGETHER (in assembly)
      asm volatile(
        "in   r16, %[port]      \n\t"  // load current PORTB value
into register r16.
        "ldi  r17, %[mask]      \n\t"  // load mask for PB1, PB2, PB3
into r17
        "eor  r16, r17          \n\t"  // flip those bits- if they
were 0 → become 1, if they were 1 → become 0
        "out  %[port], r16      \n\t"  // write updated bits back to
PORTB.
        :
        : [port] "I" (_SFR_IO_ADDR(PORTB)),
          [mask] "I" (LED_MASK)
        : "r16", "r17"
      );
      break;

    case 1: {  // Mode 1: chasing pattern 9 -> 10 -> 11
      uint8_t newPort = PORTB & ~LED_MASK;  // Clears the LED bits
(PB1–PB3) while leaving other PORTB bits untouched

      if (stepIndex == 0) {  // If stepIndex is 0 → turn on PB1
        newPort |= (1 << PB1);
      } else if (stepIndex == 1) { // If stepIndex is 1 → turn on PB2
        newPort |= (1 << PB2);
      } else { // stepIndex == 2 // If stepIndex is 2 → turn on PB3
        newPort |= (1 << PB3);
      }
```

```
      stepIndex = (stepIndex + 1) % 3;
      PORTB = newPort; // updates the port
      break;
    }

    case 2: {  // Mode 2: 3-bit binary counter on LEDs
      uint8_t newPort = PORTB & ~LED_MASK;  // clear LED bits
      uint8_t pattern = stepIndex & 0x07;   // 0..7

      if (pattern & 0x01) newPort |= (1 << PB1);
      if (pattern & 0x02) newPort |= (1 << PB2);
      if (pattern & 0x04) newPort |= (1 << PB3);

      stepIndex = (stepIndex + 1) & 0x07;  // increments and wraps at
0 – 7
      PORTB = newPort; //Result: LEDs display the binary count pattern
000, 001, 010, ..., 111 over and over.
      break;
    }
  }
}

// ------------------------------ setup ------------------------------
void setup() {
  // LEDs: PB1, PB2, PB3 (pins 9,10,11) as outputs
  DDRB |= (1 << DDB1) | (1 << DDB2) | (1 << DDB3); //DDRB controls
direction of Port B pins
  // Setting bits DDB1, DDB2, DDB3 = 1 → pins 9, 10, 11 are outputs

  // Button: PD2 (pin 2) as input with pull-up
  DDRD &= ~(1 << DDD2);    // input
  PORTD |= (1 << PORTD2);  // enable internal pull-up

  // Reset Timer1
  TCCR1A = 0; // TCCR1A, TCCR1B = 0 → off and in normal mode initially
  TCCR1B = 0;
  TCNT1  = 0; //TCNT1 = 0 resets count to zero.

  // CTC mode
  TCCR1B |= (1 << WGM12); // Sets WGM12 = 1 → Timer1 in CTC mode
(“Clear Timer on Compare Match”)
// Timer counts up from 0 to OCR1A, then:
// triggers interrupt
// resets back to 0.

  // Start in slow mode (mode 0)
  OCR1A = OCR_SLOW;

  // Enable Timer1 Compare A interrupt
```

```
  TIMSK1 |= (1 << OCIE1A);

  // Start Timer1 with prescaler 1024
  TCCR1B |= (1 << CS12) | (1 << CS10);

  // Enable global interrupts
  sei();
}

// ------------------------------ loop ------------------------------
void loop() {
  // Button edge detect (HIGH -> LOW = press) with simple debounce
  static uint8_t lastState = 1;  // 1 = HIGH (not pressed)
  uint8_t reading = (PIND & (1 << PIND2)) ? 1 : 0;

  if (reading != lastState) {
    delay(10); // debounce
    reading = (PIND & (1 << PIND2)) ? 1 : 0;

    if (lastState == 1 && reading == 0) {
      // Button press: cycle mode
      mode = (mode + 1) % 3;
      stepIndex = 0;  // restart pattern

      uint16_t newOCR;
      if (mode == 0) {
        newOCR = OCR_SLOW;   // all blink slowly
      } else if (mode == 1) {
        newOCR = OCR_MED;    // chase medium speed
      } else {
        newOCR = OCR_FAST;   // binary fast
      }

      cli();
      OCR1A = newOCR;
      sei();
    }
    lastState = reading;
  }
}
```