MACQUARIE
University
SYDNEY · AUSTRALIA

Bit Logic and Interrupts

ELEC3042 EMBEDDED SYSTEMS

Lecture 2

Recap – Setting up inputs & outputs

Register	Input	Output
Data Direction (DDRB, DDRC, DDRD)	Write 0	Write 1
Port (PORTB, PORTC, PORTD)	Write 1 for pull-up	Value for output
Input (PINB, PINC, PIND)	Read value of input	

Assignment	Usage
|=	Assign a 1 and leave other bits unchanged
&=	Assign a 0 and leave other bits unchanged

Setting multiple bits at once

THESE ARE ALL EQUIVALENT

- set pin 2 & 5 as output, the rest as inputs
- set output LOW on pin 2 & 5, pull-up resistors on input

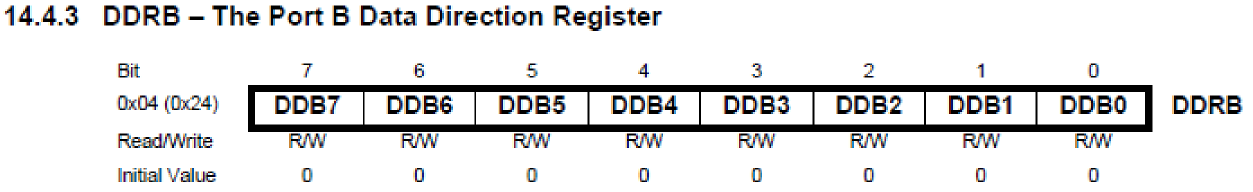
```
DDRB  = 0b00100100;
PORTB = 0b11011011;

DDRB  = 0x24;
PORTB = 0xDB;

DDRB  = 36;
PORTB = 219;
```

Macros for setting bits

THESE ARE ALL EQUIVALENT

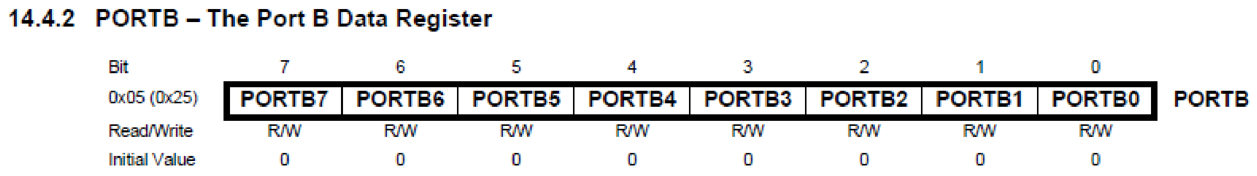
- set pin 2 & 5 as output
- set output LOW on pin 2 & 5

```
DDRB  |= 0b00100100;
PORTB &= 0b11011011;

DDRB  |= _BV(DDB2) | _BV(DDB5);
PORTB &= ~_BV(PORTB2) & ~_BV(PORTB5);

DDRB  |= (1<<DDB2) | (1<<DDB5);
PORTB &= ~(1<<PORTB2) & ~(1<<PORTB5);
```



14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Exercise

- Setup Port B pins 0, 1, 2, 3 as input, and pins 4, 5, 6, 7 as output
- Make sure all inputs have pull-up resistors active
- Set pins 4 and 5 to have an initial value of 0b0, and pins 6 and 7 to have an initial value of 0b1

Reading bits

- Want to know whether a particular bit in a register is a 1 or 0
- Create a “mask” that puts a 1 in the bit position of interest and set the rest of the bits to 0
- AND register and mask together to see if it equals 0

Example: Test if a button connected to PORTC Pin 2 is pressed
- If pressed, the output should be 0
- If not pressed, the output will not be 0

Pressed

PINC	X	X	X	X	X	0	X	X
MASK	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	0	0

&

=0

Not Pressed

PINC	X	X	X	X	X	1	X	X
MASK	0	0	0	0	0	1	0	0
	0	0	0	0	0	1	0	0

&

=4

Code

```
// setup
DDRB  |= 0b00100000;                    // PORTB pin 5 (D13) output
PORTB &= 0b11011111;                    // turn off LED
DDRC  &= 0b11111011;                    // set PORT C, Pin 2 as input
PORTC |= 0b00000100;                    // pull up PORT C, Pin 2

while (1) {
    if ((PINC & 0b00000100) == 0) {     // button is pressed
        PORTB |= 0b00100000;            // turn on LED
    } else {                            // button not pressed
        PORTB &= 0b11011111;            // turn off LED
    }
}
```

Mask

Reading multiple inputs

```
// setup
DDRC  &= 0b11111001;                    // set PORT C, Pin 1 & 2 as input
PORTC |= 0b00000110;                    // pull up PORT C, Pin 1 & 2

while (1) {
    if ((PINC & 0b00000110) == 0) {               // both buttons are pressed
       // do A
    } else if ((PINC & 0b00000100) == 0) {        // button on Pin 2 pressed
       // do B
    } else if ((PINC & 0b00000010) == 0) {        // button on Pin 1 pressed
       // do C
    } else {                                      // buttons not pressed
       // do D
    }
}
```

Reading multiple inputs

```
// setup
DDRC  &= 0b11111001;                    // set PORT C, Pin 1 & 2 as input
PORTC |= 0b00000110;                    // pull up PORT C, Pin 1 & 2

while (1) {
    if ((PINC & 0b00000110) == 0) {
       // do A
    } else if ((PINC & 0b00000100) == 0) {
       // do B
    } else if ((PINC & 0b00000010) == 0) {
       // do C
    } else {
       // do D
    }
}
```

Both Pressed

PINC	X	X	X	X	X	0	0	X
MASK	0	0	0	0	0	1	1	0
	0	0	0	0	0	0	0	0

&

=0

Only PC1 Pressed

PINC	X	X	X	X	X	1	0	X
MASK	0	0	0	0	0	1	1	0
	0	0	0	0	0	1	0	0

&

≠0

Example: LED Morse Code

```
DDRB |= 0b00100000;                   // PORTB pin 5 (D13) output
PORTB &= 0b11011111;                  // turn off LED

uint32_t sos = 0b0100100100111011101110010010010; // SOS(... --- ...)
uint32_t mask = 0x80000000;           // used to read one bit at a time

while (1) {
    if ((sos & mask) == 0) {          // test whether current bit is 0
        PORTB &= ~_BV(PB5);
    } else {
        PORTB |= _BV(PB5);
    }
    delay_ms(400);
    mask = mask >> 1;                 // shift mask to next bit
    if (mask == 0) {
        mask = 0x80000000;            // reset mask
    }
}
```

Example: LED Morse Code

```
DDRB  |= 0b00100000;                      // PORTB pin 5 (D13) output
PORTB &= 0b11011111;                      // turn off LED

uint32_t sos = 0b01001001001110111011100100100010; // SOS(... --- ...)
uint32_t mask = 0x80000000;

while (1) {
    if ((sos & mask) == 0) {
        PORTB &= ~_BV(PB5);
    } else {
        PORTB |= _BV(PB5);
    }
    delay_ms(400);
    mask = mask >> 1;                     // shift mask to next bit
    if (mask == 0) {
        mask = 0x80000000;                // reset mask
    }
}
```

Iteration 1

sos	0	1	0	0	1	0	0	1	…
mask	1	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0

&

=0

Example: LED Morse Code

```
DDRB  |= 0b00100000;                    // PORTB pin 5 (D13) output
PORTB &= 0b11011111;                    // turn off LED

uint32_t sos = 0b01001001001110111011100100100010; // SOS(... --- ...)
uint32_t mask = 0x80000000;

while (1) {
    if ((sos & mask) == 0) {
        PORTB &= ~_BV(PB5);
    } else {
        PORTB |= _BV(PB5);
    }
    delay_ms(400);
    mask = mask >> 1;
    if (mask == 0) {
        mask = 0x80000000;
    }
}
```

Iteration 1

sos	0	1	0	0	1	0	0	1	…
mask	1	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0

&

=0

Iteration 2

sos	0	1	0	0	1	0	0	1	…
mask	0	1	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0

&

≠0

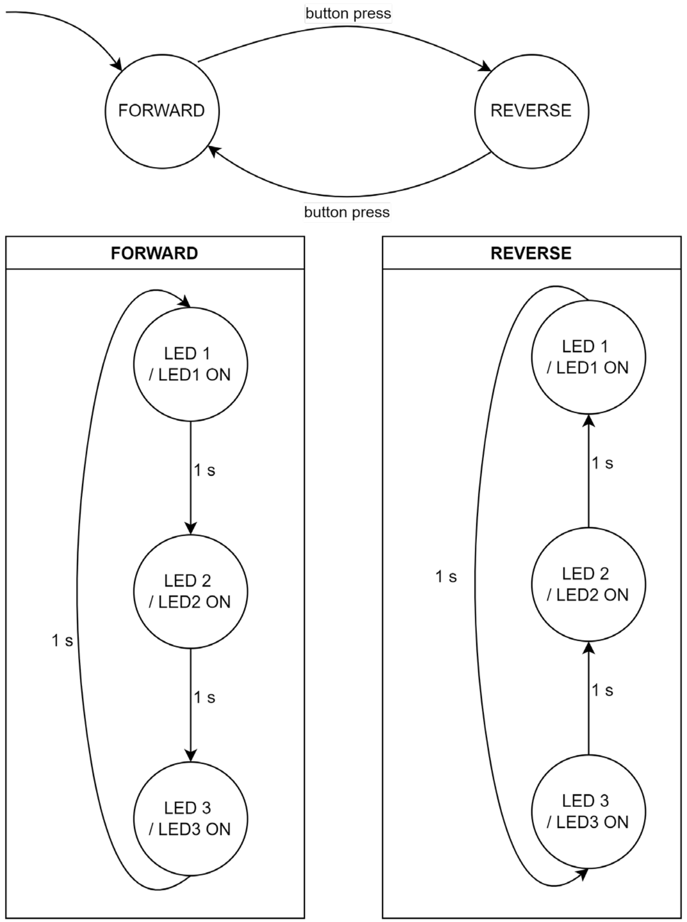
Interrupts

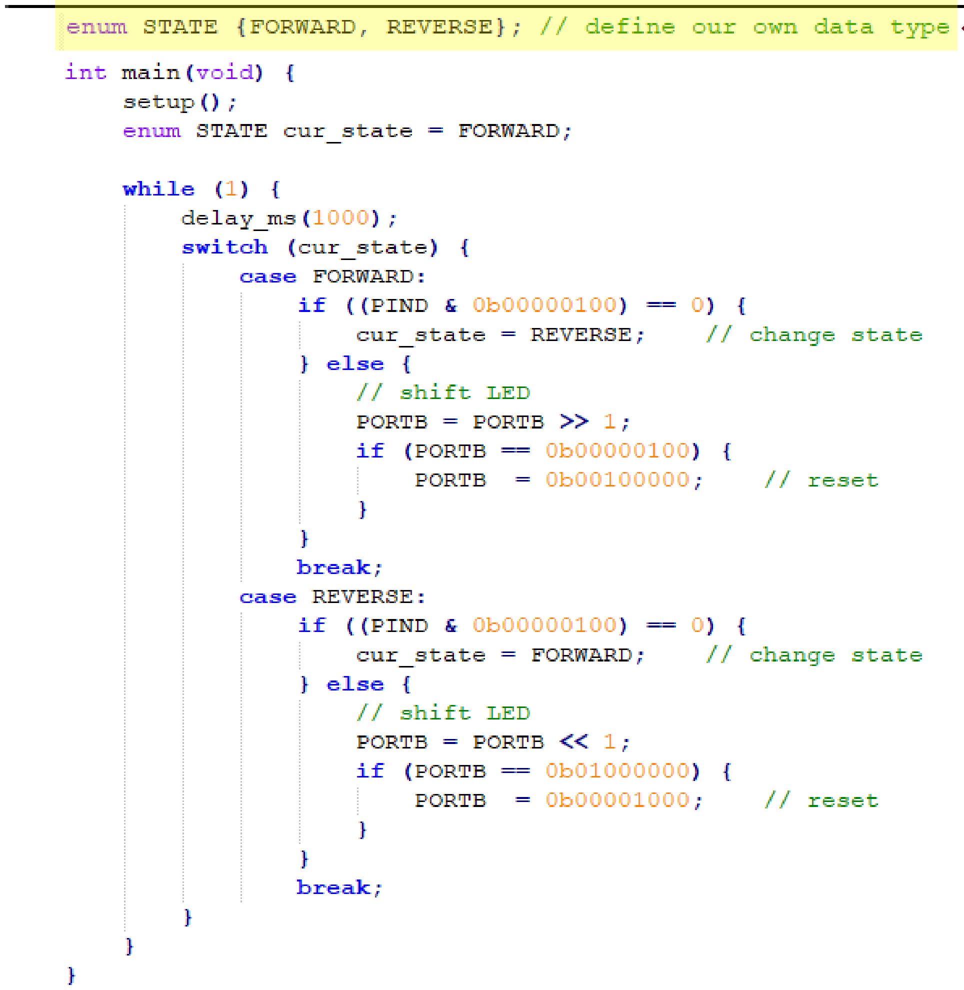
Toy Problem

Create a program that cycles and lights up one of three LEDs for one second each. A push button is used to change the direction of the cycling whenever it is pressed.

DESIGN:

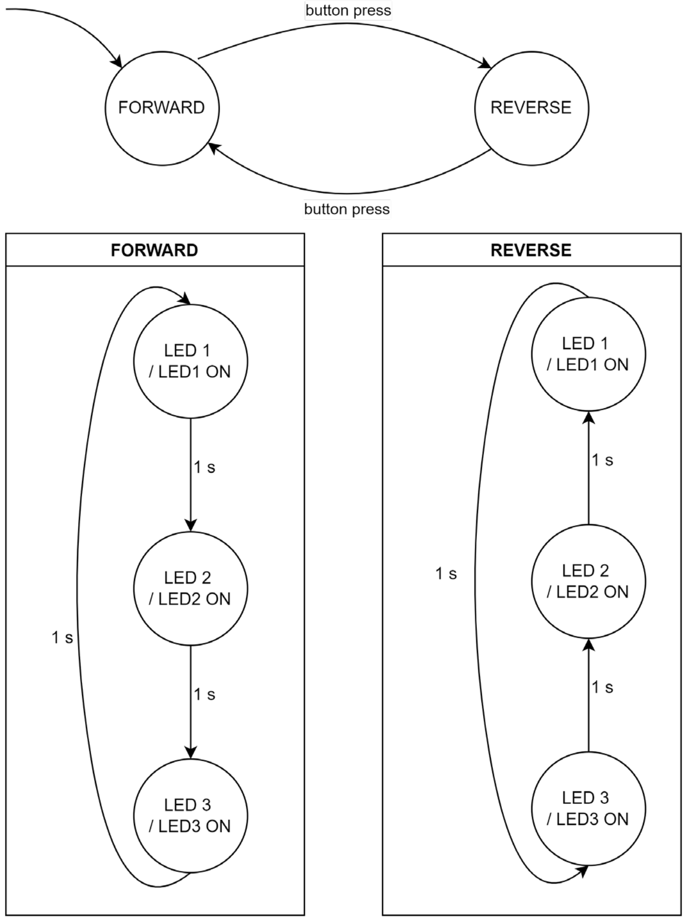
- LEDs are connected to PB5 (LED1), PB4 (LED2), PB3 (LED3)
- Pushbutton is connected to PD2

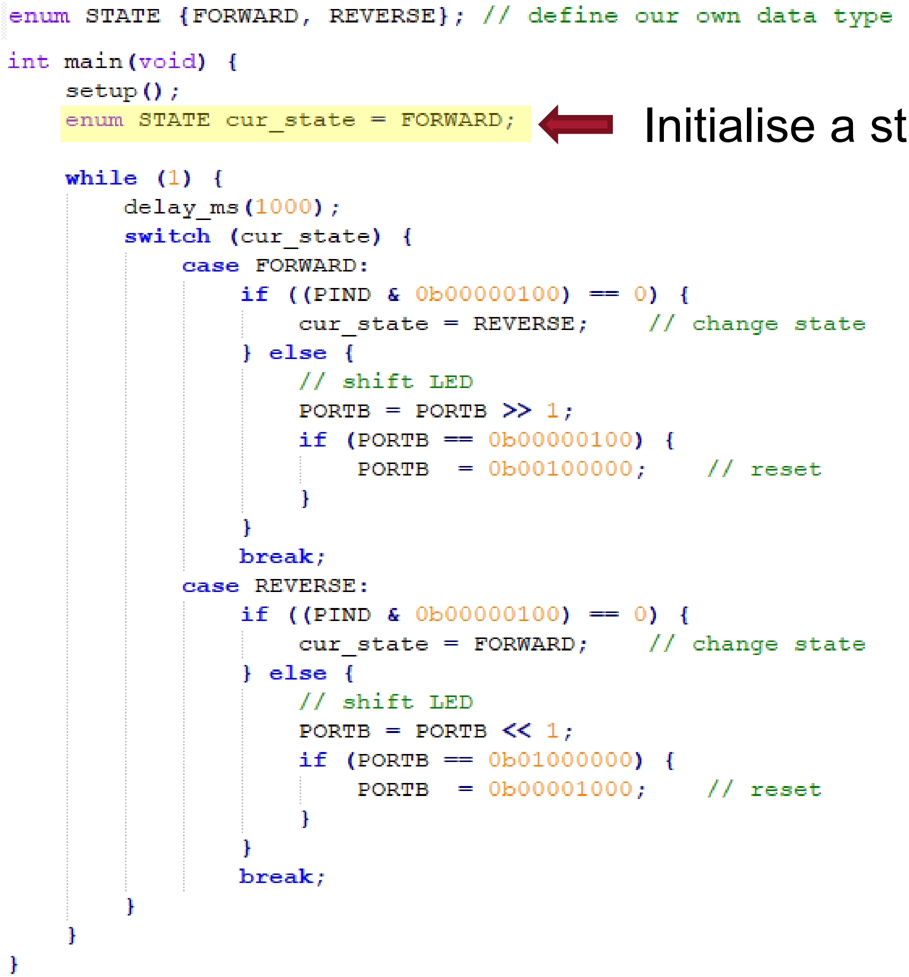




```
enum STATE {FORWARD, REVERSE}; // define our own data type

int main(void) {
    setup();
    enum STATE cur_state = FORWARD;

    while (1) {
        delay_ms(1000);
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = REVERSE;     // change state
                } else {
                    // shift LED
                    PORTB = PORTB >> 1;
                    if (PORTB == 0b00000100) {
                        PORTB  = 0b00100000;     // reset
                    }
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;     // change state
                } else {
                    // shift LED
                    PORTB = PORTB << 1;
                    if (PORTB == 0b01000000) {
                        PORTB  = 0b00001000;     // reset
                    }
                }
                break;
        }
    }
}
```

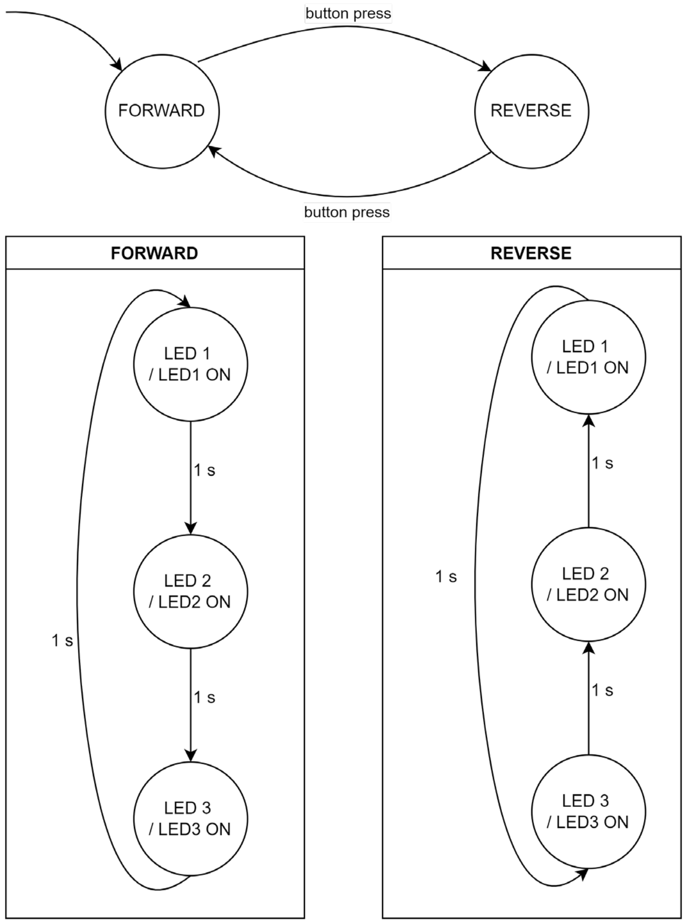
Makes code easier to read

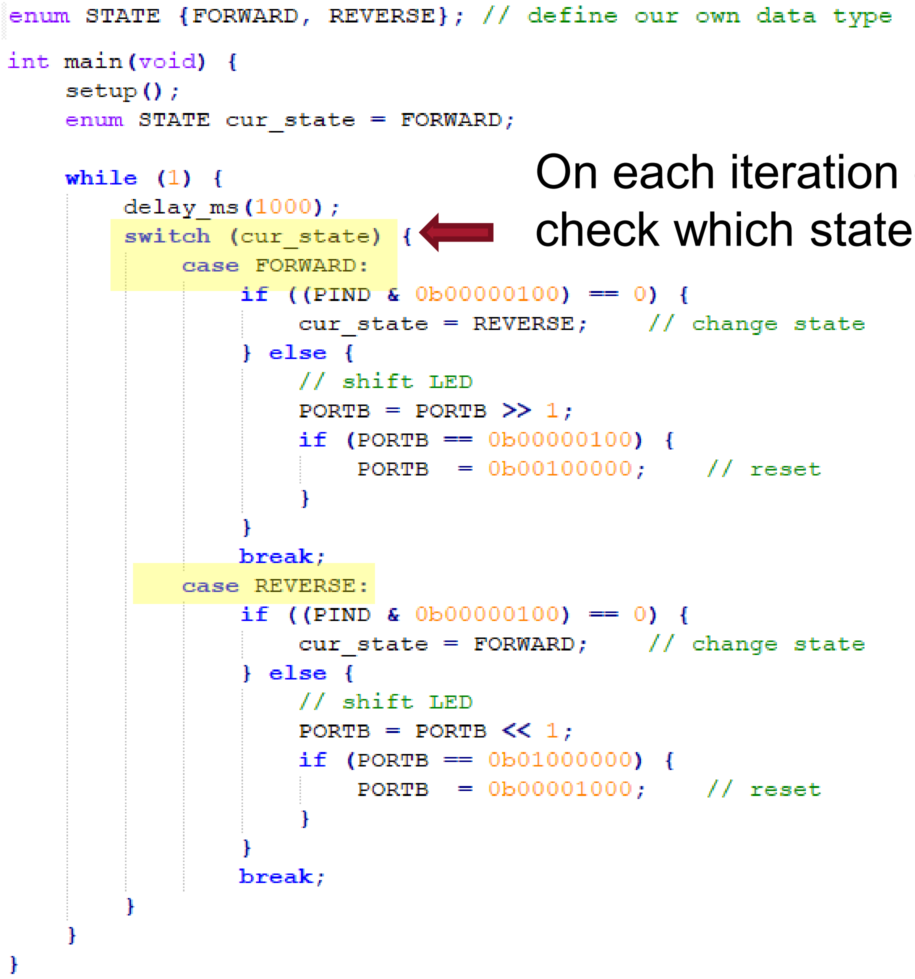




```
enum STATE {FORWARD, REVERSE}; // define our own data type

int main(void) {
    setup();
    enum STATE cur_state = FORWARD;

    while (1) {
        delay_ms(1000);
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = REVERSE;    // change state
                } else {
                    // shift LED
                    PORTB = PORTB >> 1;
                    if (PORTB == 0b00000100) {
                        PORTB  = 0b00100000;    // reset
                    }
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;    // change state
                } else {
                    // shift LED
                    PORTB = PORTB << 1;
                    if (PORTB == 0b01000000) {
                        PORTB  = 0b00001000;    // reset
                    }
                }
                break;
        }
    }
}
```

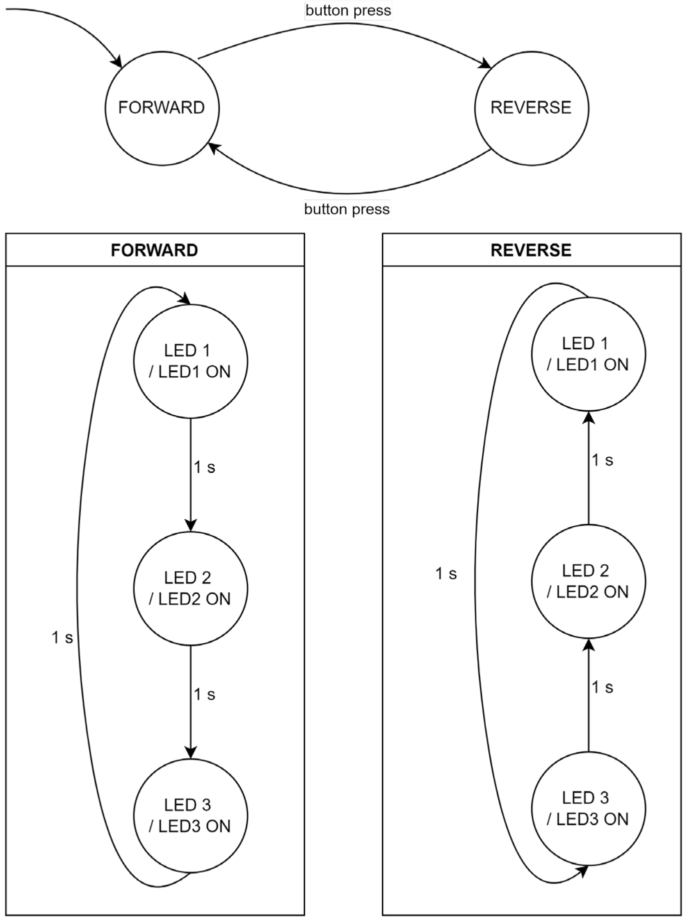
Initialise a starting state

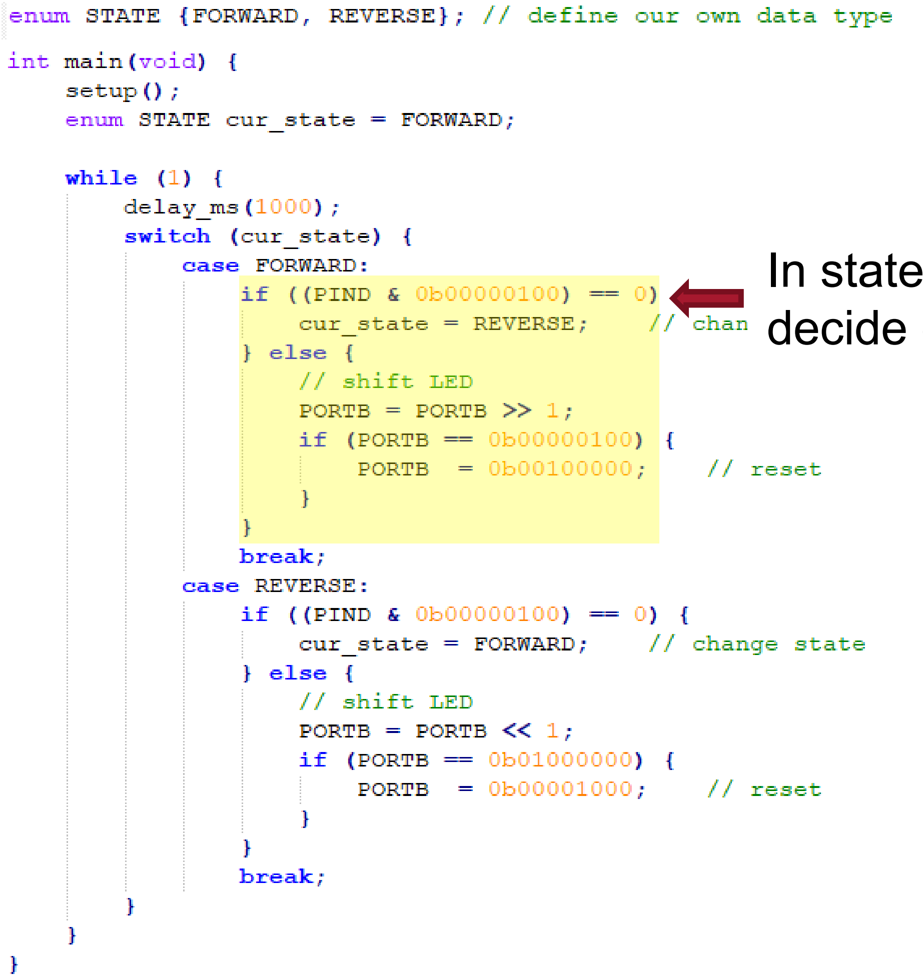




```
enum STATE {FORWARD, REVERSE}; // define our own data type

int main(void) {
    setup();
    enum STATE cur_state = FORWARD;

    while (1) {
        delay_ms(1000);
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = REVERSE;     // change state
                } else {
                    // shift LED
                    PORTB = PORTB >> 1;
                    if (PORTB == 0b00000100) {
                        PORTB  = 0b00100000;     // reset
                    }
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;     // change state
                } else {
                    // shift LED
                    PORTB = PORTB << 1;
                    if (PORTB == 0b01000000) {
                        PORTB  = 0b00001000;     // reset
                    }
                }
                break;
        }
    }
}
```

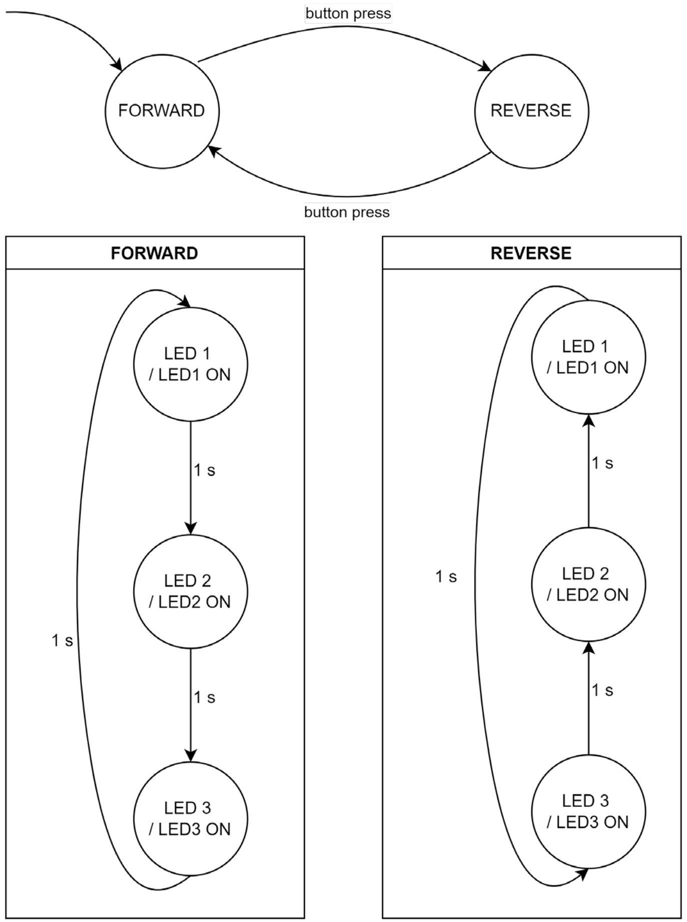
On each iteration of loop, check which state we are in

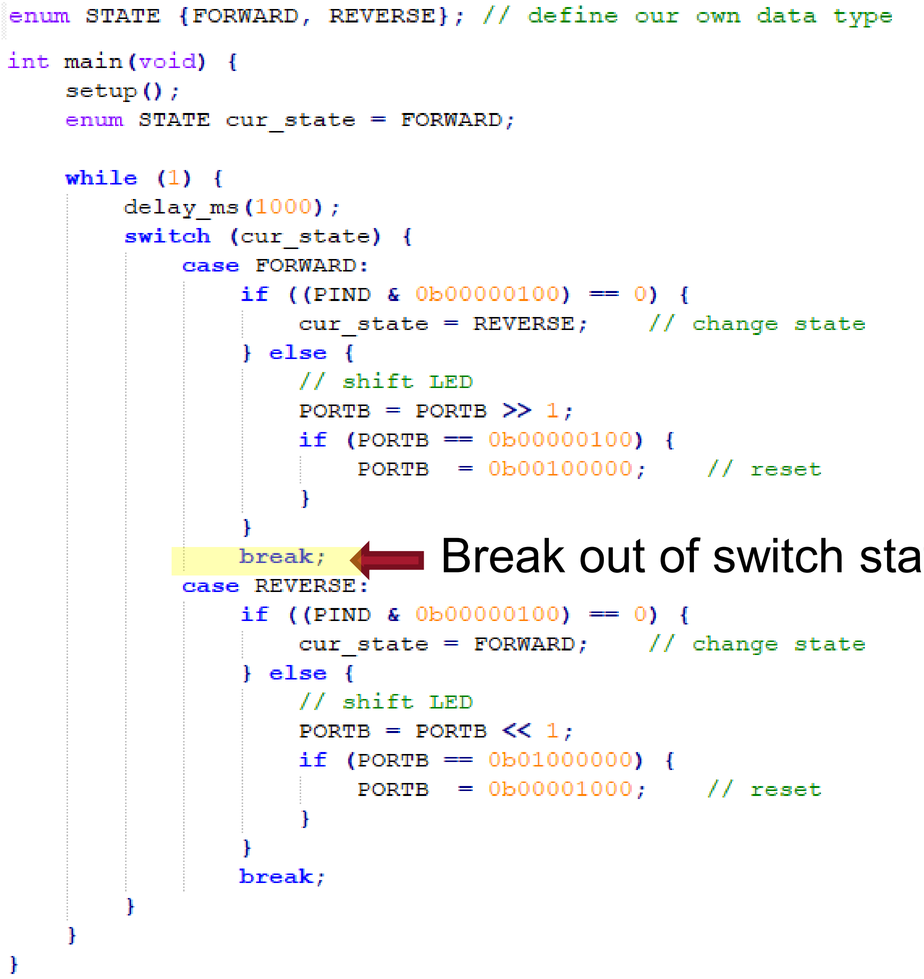




```
enum STATE {FORWARD, REVERSE}; // define our own data type

int main(void) {
    setup();
    enum STATE cur_state = FORWARD;

    while (1) {
        delay_ms(1000);
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0)
                    cur_state = REVERSE;     // chan
                } else {
                    // shift LED
                    PORTB = PORTB >> 1;
                    if (PORTB == 0b00000100) {
                        PORTB  = 0b00100000;     // reset
                    }
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;     // change state
                } else {
                    // shift LED
                    PORTB = PORTB << 1;
                    if (PORTB == 0b01000000) {
                        PORTB  = 0b00001000;     // reset
                    }
                }
                break;
        }
    }
}
```

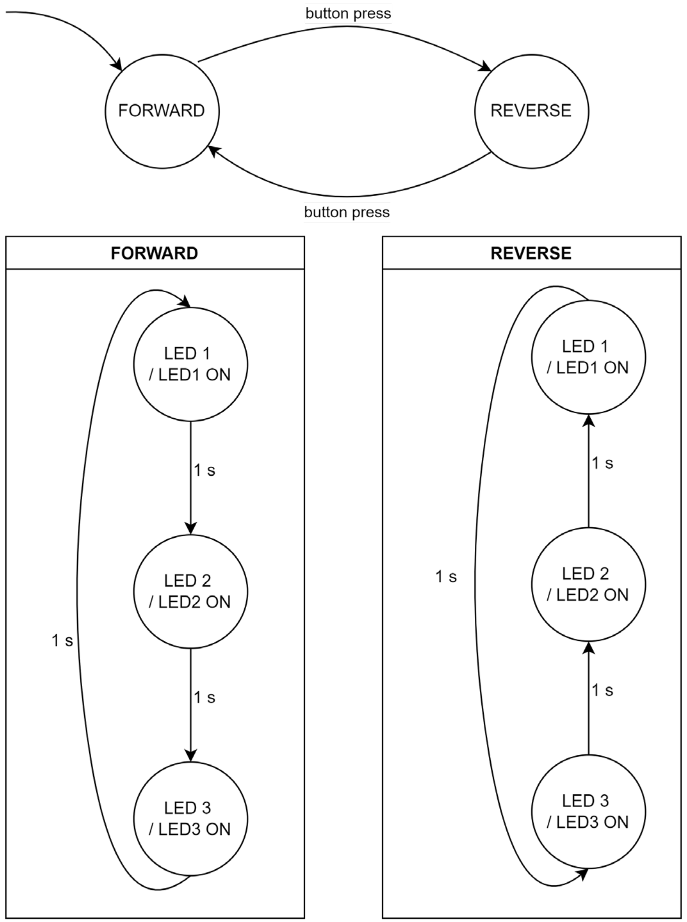
In state, check input to decide on what to do

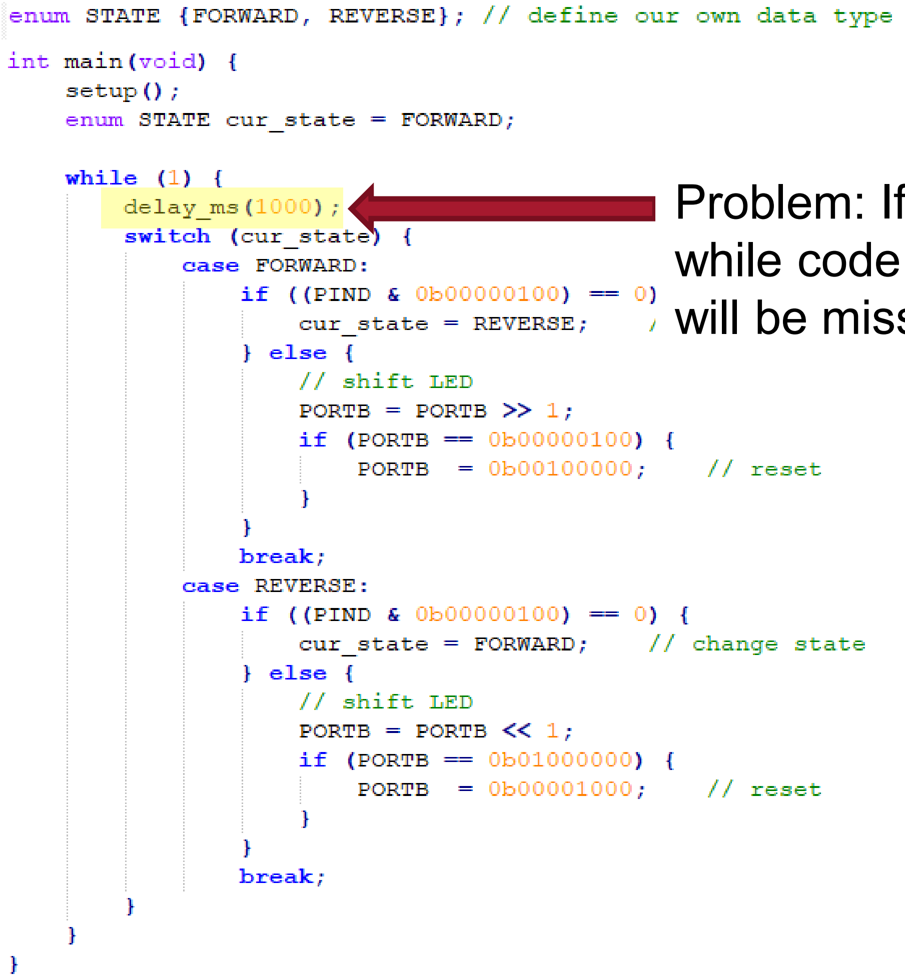




```
enum STATE {FORWARD, REVERSE}; // define our own data type

int main(void) {
    setup();
    enum STATE cur_state = FORWARD;

    while (1) {
        delay_ms(1000);
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = REVERSE;     // change state
                } else {
                    // shift LED
                    PORTB = PORTB >> 1;
                    if (PORTB == 0b00000100) {
                        PORTB  = 0b00100000;     // reset
                    }
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;     // change state
                } else {
                    // shift LED
                    PORTB = PORTB << 1;
                    if (PORTB == 0b01000000) {
                        PORTB  = 0b00001000;     // reset
                    }
                }
                break;
        }
    }
}
```

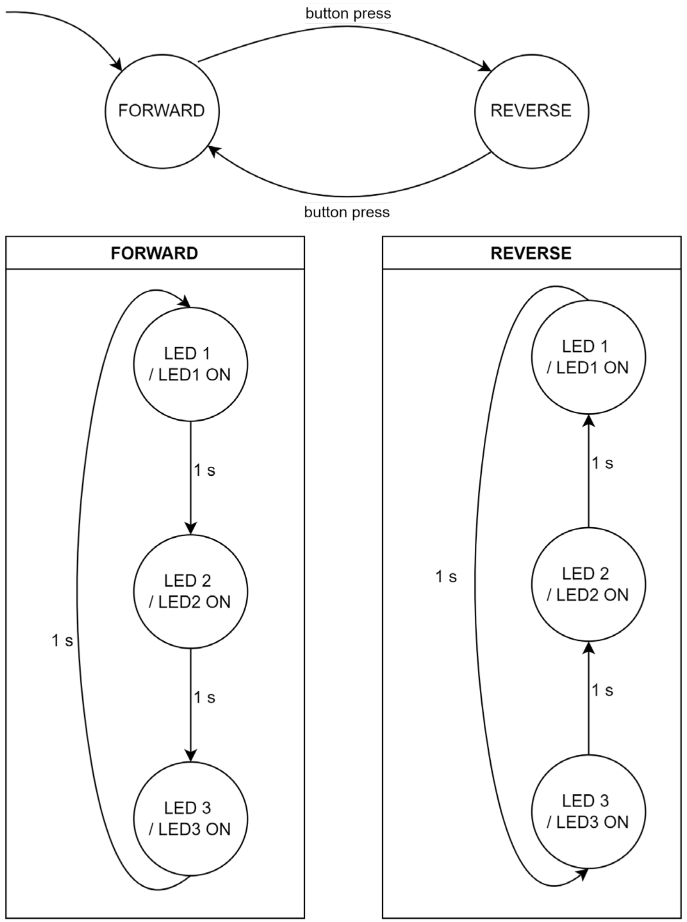
Break out of switch statement





```
enum STATE {FORWARD, REVERSE}; // define our own data type

int main(void) {
    setup();
    enum STATE cur_state = FORWARD;

    while (1) {
        delay_ms(1000);
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0)
                    cur_state = REVERSE;     /
                } else {
                    // shift LED
                    PORTB = PORTB >> 1;
                    if (PORTB == 0b00000100) {
                        PORTB  = 0b00100000;    // reset
                    }
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;    // change state
                } else {
                    // shift LED
                    PORTB = PORTB << 1;
                    if (PORTB == 0b01000000) {
                        PORTB  = 0b00001000;    // reset
                    }
                }
                break;
        }
    }
}
```

Problem: If button is pressed while code execution is here, it will be missed



Interrupts

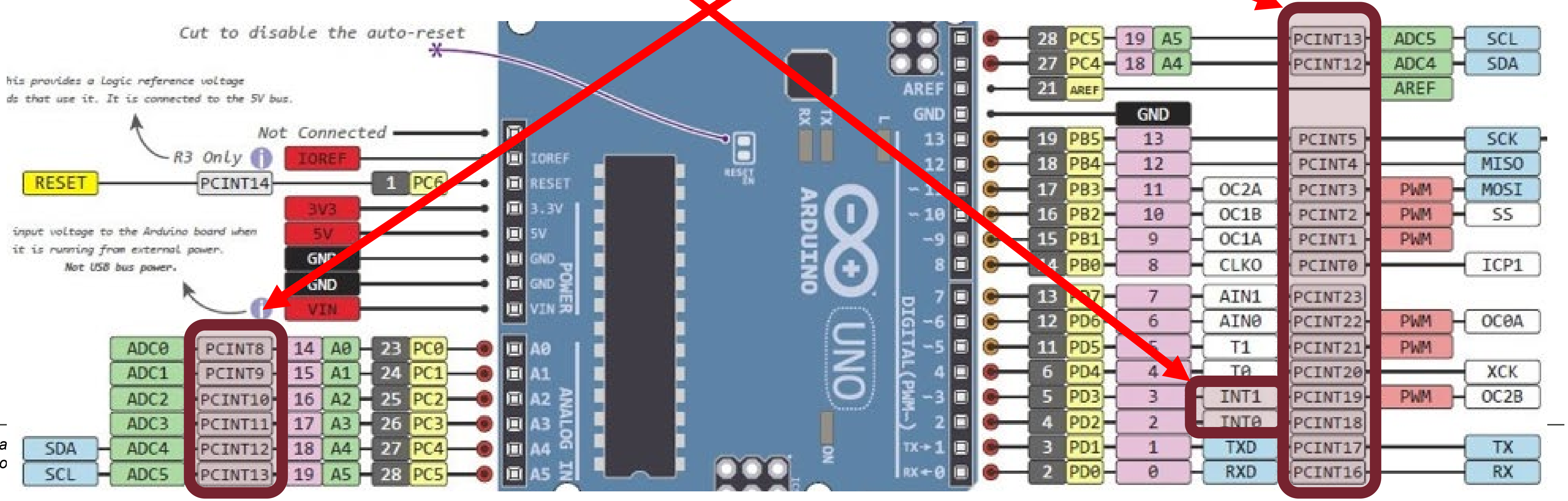
- Temporarily delay execution of current code while some more urgent code is run on the CPU
- Is used to indicate an important external or internal event in a system
- Frees us from needing to constantly check for an event

Sources of Interrupts

- Port pins
- Timers
- UART, SPI, I2C
- ADC
- Analog Comparator
- EEPROM

Port Pin Interrupts

- Allows generation of interrupts from external peripherals and events
- Each pin has an associated pin change interrupt (PCINT0:23)
- Two high-priority interrupts (INT0, INT1)

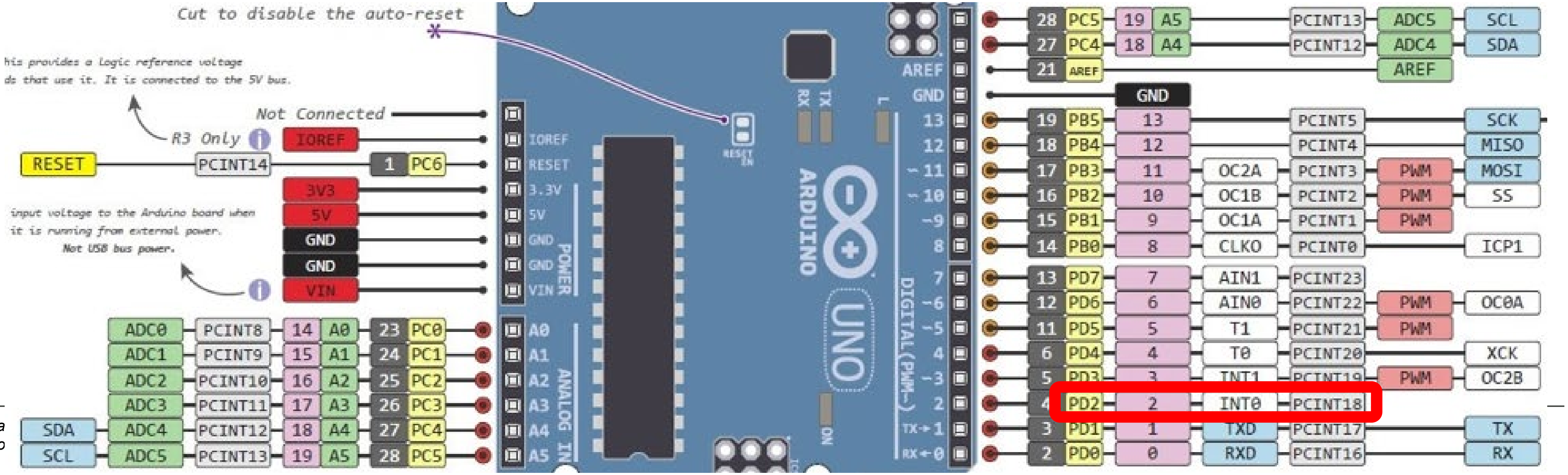


Cut to disable the auto-reset
his provides a logic reference voltage
ds that use it. It is connected to the 5V bus.
Not Connected
R3 Only
IOREF
RESET
PCINT14
1 PC6
3V3
5V
GND
GND
VIN
input voltage to the Arduino board when
it is running from external power.
Not USB bus power.
ADC0 PCINT8 14 A0 23 PC0
ADC1 PCINT9 15 A1 24 PC1
ADC2 PCINT10 16 A2 25 PC2
ADC3 PCINT11 17 A3 26 PC3
SDA ADC4 PCINT12 18 A4 27 PC4
SCL ADC5 PCINT13 19 A5 28 PC5
IOREF
RESET
3.3V
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
RESET EN
RX
TX
L
ARDUINO
UNO
ON
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
28 PC5 19 A5 PCINT13 ADC5 SCL
27 PC4 18 A4 PCINT12 ADC4 SDA
21 AREF AREF
GND
19 PB5 13 PCINT5 SCK
18 PB4 12 PCINT4 MISO
17 PB3 11 OC2A PCINT3 PWM MOSI
16 PB2 10 OC1B PCINT2 PWM SS
15 PB1 9 OC1A PCINT1 PWM
14 PB0 8 CLKO PCINT0 ICP1
13 PD7 7 AIN1 PCINT23
12 PD6 6 AIN0 PCINT22 PWM OC0A
11 PD5 5 T1 PCINT21 PWM
6 PD4 4 T0 PCINT20 XCK
5 PD3 3 INT1 PCINT19 PWM OC2B
4 PD2 2 INT0 PCINT18
3 PD1 1 TXD PCINT17 TX
2 PD0 0 RXD PCINT16 RX

This ma
owner o

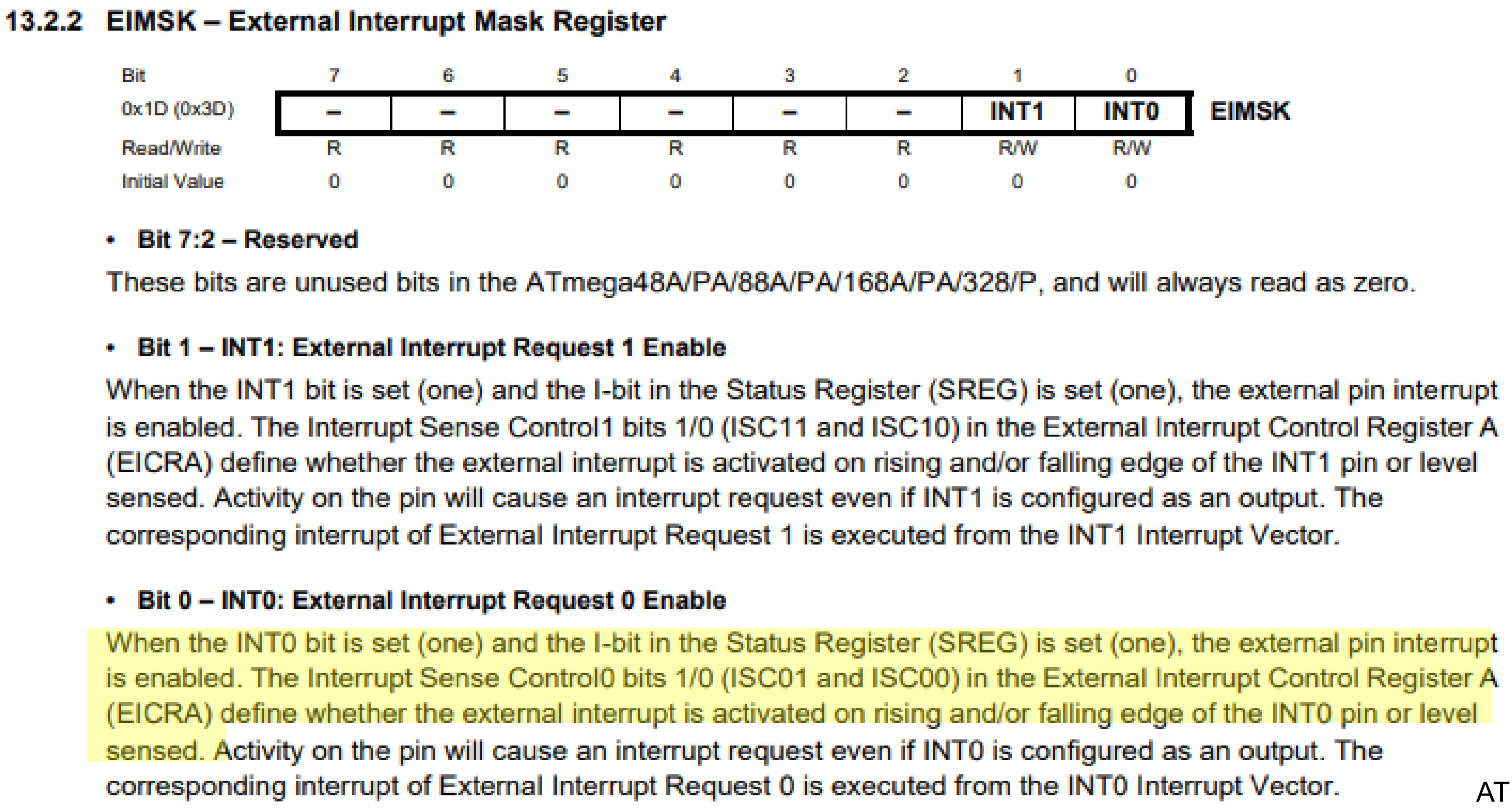
Example: Button connected to PD2

- Can choose either INT0 or PCINT18


Cut to disable the auto-reset
his provides a logic reference voltage
ds that use it. It is connected to the 5V bus.
Not Connected
R3 Only
IOREF
RESET
PCINT14
1 PC6
3V3
5V
GND
GND
VIN
input voltage to the Arduino board when
it is running from external power.
Not USB bus power.
ADC0 PCINT8 14 A0 23 PC0
ADC1 PCINT9 15 A1 24 PC1
ADC2 PCINT10 16 A2 25 PC2
ADC3 PCINT11 17 A3 26 PC3
SDA ADC4 PCINT12 18 A4 27 PC4
SCL ADC5 PCINT13 19 A5 28 PC5
IOREF RESET 3.3V 5V GND GND VIN
POWER
A0 A1 A2 A3 A4 A5
ANALOG IN
RESET EN
RX TX L
ARDUINO
UNO
ON
AREF GND 13 12 ~11 ~10 ~9 8 7 ~6 ~5 4 ~3 2 TX→1 RX←0
DIGITAL (PWM~)
28 PC5 19 A5 PCINT13 ADC5 SCL
27 PC4 18 A4 PCINT12 ADC4 SDA
21 AREF AREF
GND
19 PB5 13 PCINT5 SCK
18 PB4 12 PCINT4 MISO
17 PB3 11 OC2A PCINT3 PWM MOSI
16 PB2 10 OC1B PCINT2 PWM SS
15 PB1 9 OC1A PCINT1 PWM
14 PB0 8 CLKO PCINT0 ICP1
13 PD7 7 AIN1 PCINT23
12 PD6 6 AIN0 PCINT22 PWM OC0A
11 PD5 5 T1 PCINT21 PWM
6 PD4 4 T0 PCINT20 XCK
5 PD3 3 INT1 PCINT19 PWM OC2B
4 PD2 2 INT0 PCINT18
3 PD1 1 TXD PCINT17 TX
2 PD0 0 RXD PCINT16 RX

Example: INT0 Interrupt

1. ENABLE THE EVENT-SPECIFIC INTERRUPT



13.2.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:2 – Reserved**

These bits are unused bits in the ATmega48A/PA/88A/PA/168A/PA/328/P, and will always read as zero.

- **Bit 1 – INT1: External Interrupt Request 1 Enable**

When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 Interrupt Vector.

- **Bit 0 – INT0: External Interrupt Request 0 Enable**

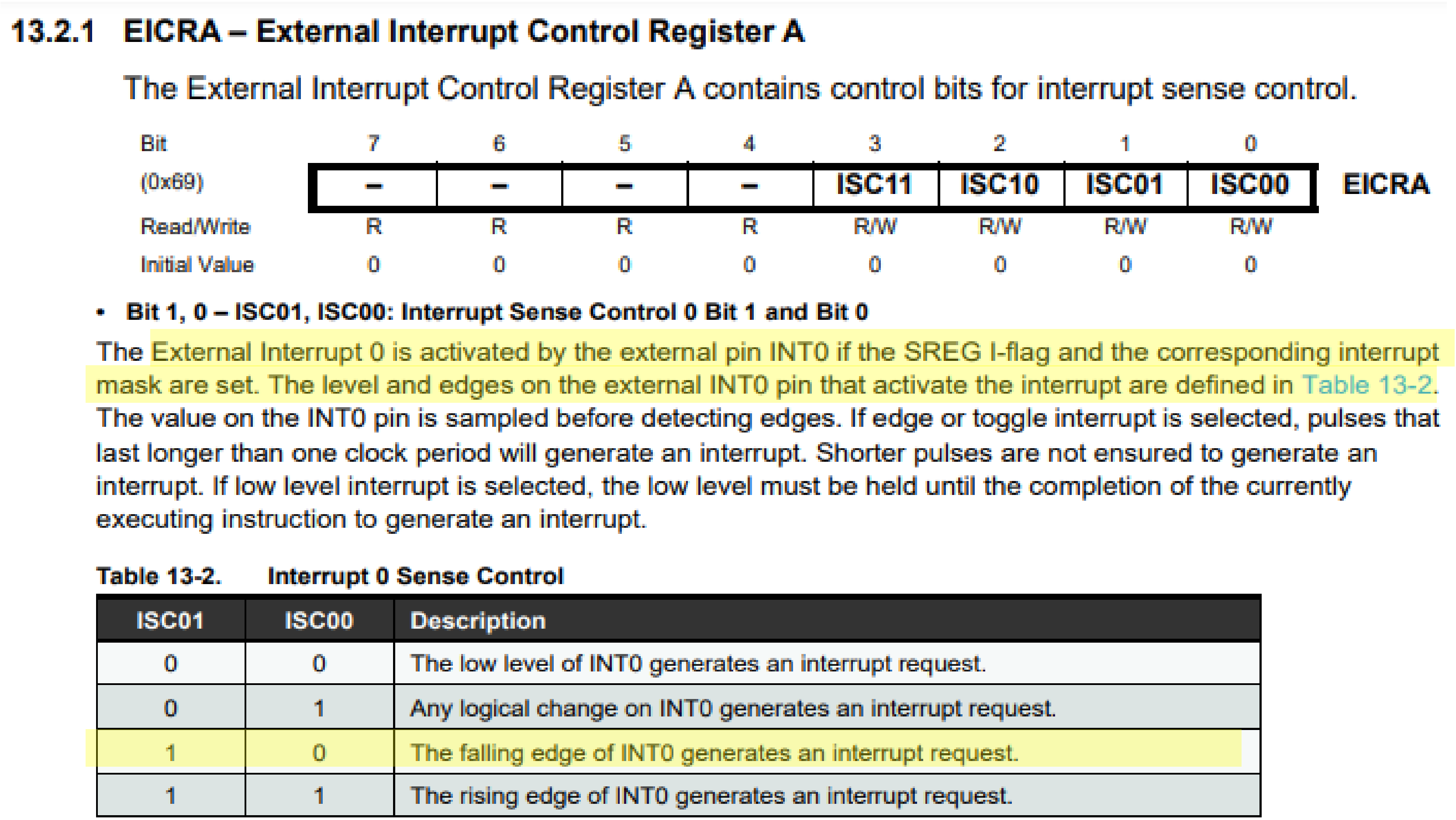
When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the External Interrupt Control Register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 Interrupt Vector.

```
EIMSK = 0b00000001;
```

ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf

Example: INT0 Interrupt

2. DEFINE HOW THE INT0 INTERRUPT SHOULD BEHAVE



13.2.1 EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1, 0 – ISC01, ISC00: Interrupt Sense Control 0 Bit 1 and Bit 0**

The External Interrupt 0 is activated by the external pin INT0 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT0 pin that activate the interrupt are defined in Table 13-2. The value on the INT0 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not ensured to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 13-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

```
EIMSK = 0b00000001;
EICRA = 0b00000010;
```

ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf

When does an interrupt occur?

When 3 things are satisfied:

1. An event has been recorded
2. The event-specific interrupt enable has been set
3. The global interrupt enable flag has been set

```
EIMSK = 0b00000001;
EICRA = 0b00000010;
sei(); // enable interrupts
```

What happens when an interrupt occurs?

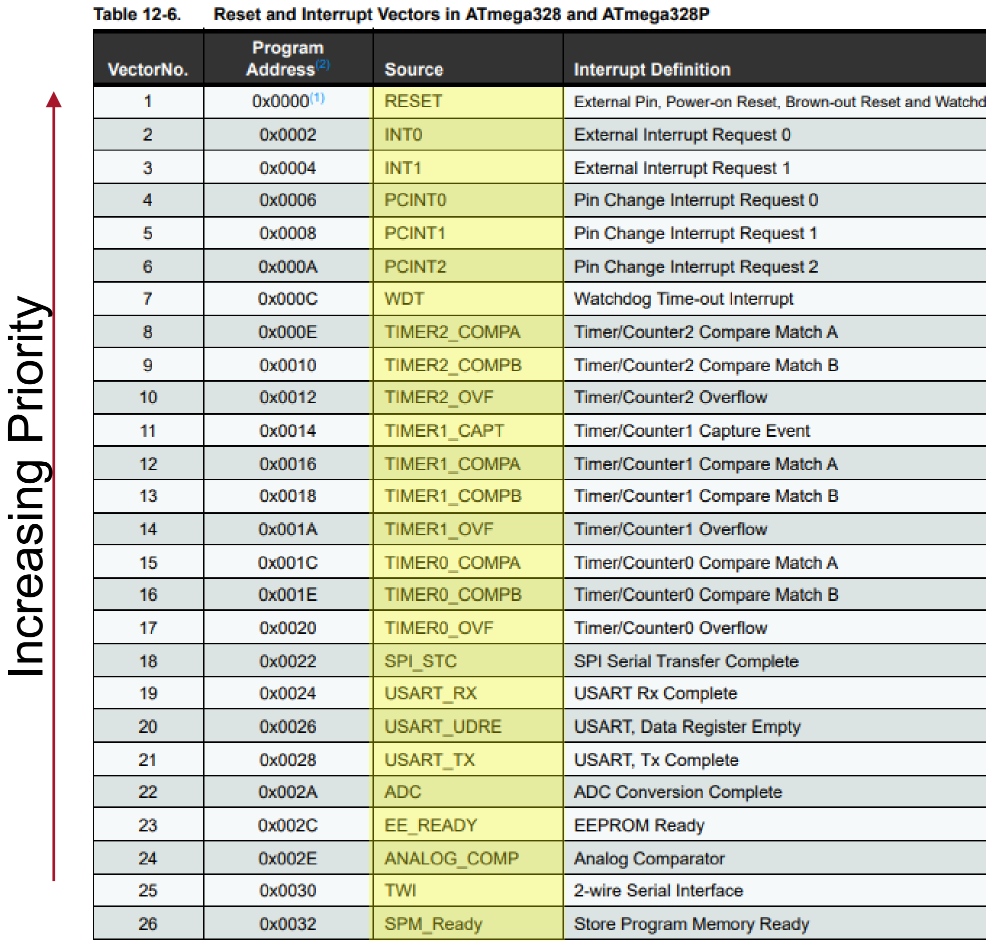
1. Current program execution is stopped
2. Context is saved (register values, program counter)
3. Control jumps to Interrupt Service Routine (ISR)

Interrupt Service Routine (ISR)

- Code that gets run when interrupt is raised
- All ISR have the same template:

```
ISR(SOURCE_vect) {
    // ISR code here
}
```

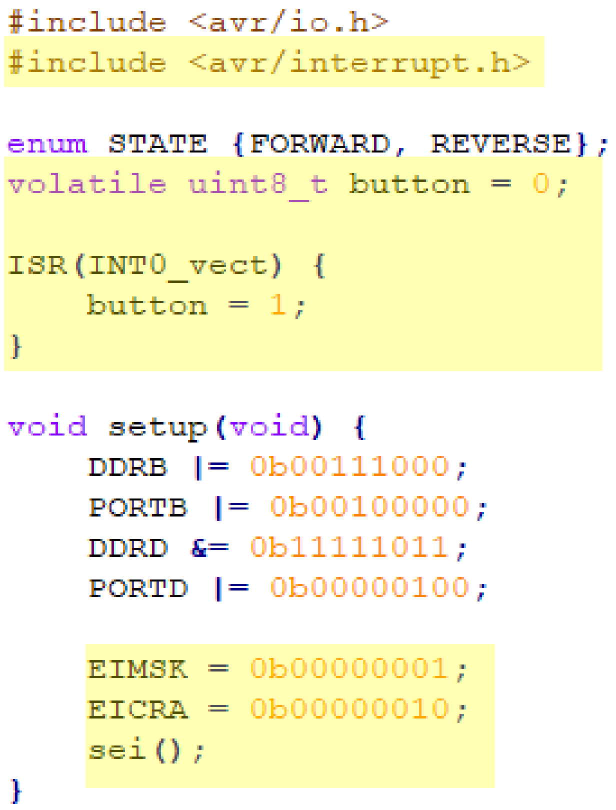
- Want to do minimum amount of work in ISR



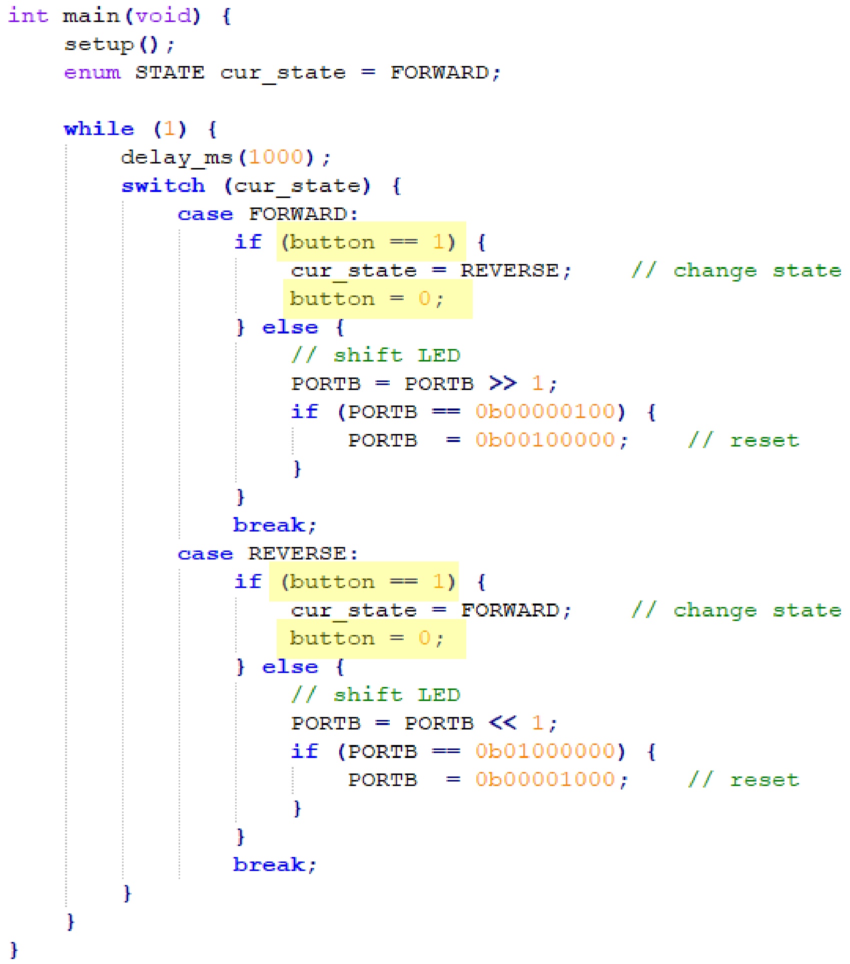
Increasing Priority

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchd
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1_COMPB	Timer/Counter1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0_OVF	Timer/Counter0 Overflow
18	0x0022	SPI_STC	SPI Serial Transfer Complete
19	0x0024	USART_RX	USART Rx Complete
20	0x0026	USART_UDRE	USART, Data Register Empty
21	0x0028	USART_TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE_READY	EEPROM Ready
24	0x002E	ANALOG_COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM_Ready	Store Program Memory Ready



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

enum STATE {FORWARD, REVERSE};
volatile uint8_t button = 0;

ISR(INT0_vect) {
    button = 1;
}

void setup(void) {
    DDRB |= 0b00111000;
    PORTB |= 0b00100000;
    DDRD &= 0b11111011;
    PORTD |= 0b00000100;

    EIMSK = 0b00000001;
    EICRA = 0b00000010;
    sei();
}
```



```c
int main(void) {
    setup();
    enum STATE cur_state = FORWARD;

    while (1) {
        delay_ms(1000);
        switch (cur_state) {
            case FORWARD:
                if (button == 1) {
                    cur_state = REVERSE;     // change state
                    button = 0;
                } else {
                    // shift LED
                    PORTB = PORTB >> 1;
                    if (PORTB == 0b00000100) {
                        PORTB  = 0b00100000;     // reset
                    }
                }
                break;
            case REVERSE:
                if (button == 1) {
                    cur_state = FORWARD;     // change state
                    button = 0;
                } else {
                    // shift LED
                    PORTB = PORTB << 1;
                    if (PORTB == 0b01000000) {
                        PORTB  = 0b00001000;     // reset
                    }
                }
                break;
        }
    }
}
```

Interrupt Summary

- Choose an appropriate interrupt
- Configure appropriate interrupt registers
- Enable specific interrupt bit
- Enable global interrupts
- Define ISR