Handling Multiple External Interrupts

Ishmaiah Julia T. Agbay, Andrea Therese A. Rodrigo, Marianne Fatima M. Valenzuela
Higher Education Department, Don Bosco Technical College Mandaluyong

***Abstract*— Microcontrollers execute instructions in a fetch-then-execute sequence based on the user-written code but it should also be prepared to handle unscheduled external and internal events. This can be done through the interrupt system found on a microcontroller. This paper discusses the handling of interrupts on the ATMEGA328P. It delves on the basics, kinds, and purpose of interrupts and the function of the Interrupt Service Routine (ISR). To show the handling of multiple external interrupts, it also presents the circuit diagrams and its corresponding codes for the basic binary counter, mainly focusing on the use of external interrupts through tactile switches. The concept of pin change is also used and shown to handle hardware interrupts and preventing button bouncing,**

I. INTRODUCTION

Interrupts are one of the most fundamental and useful principles of modern embedded processors. A microcontroller normally executes instructions in an orderly fetch then execute sequence depending on how the user would write the code, but it should also be ready to handle any unscheduled events that might occur inside or outside the microcontroller.

The AVR can do this by performing an Interrupt Service Routine (ISR). Its execution is performed by loading the beginning address of the ISR specific for the interrupt into the program counter then, the AVR processor would start running the ISR. For the ISR to function as planned, there are several conditions that it must meet. This would further be discussed along the way (see section b.4). The interrupt routine is an asynchronous process as these events occur outside the regular execution thread of the main program. It does not do things simultaneously as the regular program is stopped when the ISR runs. By default, interrupts are not interruptible, once it runs, the AVR CPU will automatically disable the Global Interrupt Enable bit to prevent the ISR from being interrupted. This is done to prevent stack overflows from too many interrupts occurring at once and to prevent the ISR from running too long.

As microcontrollers can accept inputs from I/O ports, interrupts can also be used for accepting inputs generated by external events. The type of device associated with this is referred to as the external interrupts. These are called on a given status change on the INTn pin. On the ATMEGA328P, all pins can trigger interrupts when they change the state from low to high and vice versa. Only INT0 and INT1 can be configured to trigger interrupts in a low level and it will trigger continuously until the input is no longer low. The other non-interrupt pins serve as the interrupt handlers, this allows for INT0 and INT1 to handle multiple external interrupts. The pins are grouped into three pin change interrupt vectors, if any pin changes its state, an interrupt will be triggered.

Thus, interrupts are useful as it is a way for an external or internal event to pause the current processor's activity so that it can complete a brief task before resuming where it left off. Other uses of interrupts include: serial data transfers, timer interrupts, ADC conversions, etc.

II. DISCUSSION

A. ATMega328p

The ATMega328P is a high performance, low power controller from Microchip. It is an 8-bit microcontroller based on AVR RISC architecture and it is the most popular of all the AVR controllers as it is mostly used in Arduino boards.

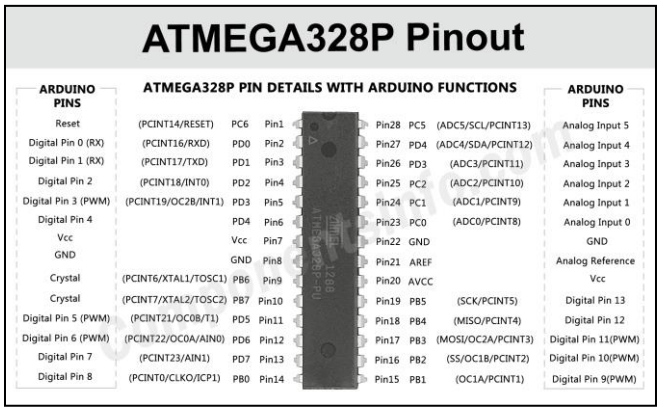

Fig. 1. ATMega328p pinout and equivalent Arduino pins

TABLE I. PIN CONFIGURATIONS AND DESCRIPTION

Pin Name	Function
PC6 (Reset)	Used as the reset pin. It can only be used as an I/O pin when the RSTDISBL fuse is programmed.
PD0 (RXD)	RXD (Data input for USART). USART serial communication interface (can be used for programming)
PD1 (TXD)	TXD (Data output for USART) USART serial communication interface (can be used for programming) INT2 (External interrupt 2 input)
PD2 (INT0)	External Interrupt source 0.
PD3 (INT1/OC2B)	External Interrupt source 1 OC2B (PWM- Timer/Counter2 Output compare Match B Output)
PD4 (XCK/T0)	T0 (Timer0 External Counter Input) XCK (USART External Clock I/O)
VCC	Connected to *positive* voltage.
GND	Connected to ground.
PB6 (XTAL1/TOSC1)	XTAL1 (Chip clock oscillator pin 1 or External clock input) TOSC1 (Timer oscillator pin 1)
PB7 (XTAL2/TOSC2)	XTAL2 (Chip clock oscillator pin 2) TOSC2 (Timer oscillator pin 2)
PD5 (T1/OC0B)	T1 (Timer1 external counter input) OC0B (PWM- Timer/Counter0 Output compare Match B Output)

PD6 (AIN0/OC0A)	AIN0 (Analog comparator positive I/P) OC0A (PWM- Timer/Counter0 Output compare Match A Output)
PD7 (AIN1)	AIN1 (Analog comparator negative I/P)
PB0 (ICP1/CLKO)	ICP1 (Timer/Counter 1 input capture pin) CLKO (Divided system clock)
PB1 (OC1A)	OC1A (Timer/Counter1 Output compare Match A Output)
PB2 (SS/OC1B)	SS (SPI Slave Select Input) OC1B (Timer/Counter1 Output compare Match B Output)
PB3 (MOSI/OC2A)	MOSI (Master Output Slave Input) OC2A (Timer/Counter2 Output compare Match A Output)
PB4 (MISO)	MISO (Master Input Slave Output)
PB5 (SCK)	SCK (SPI Bus Serial Clock)
AVCC	Power for internal ADC converter.
AREF	Analog reference pin for ADC.
GND	Ground.
PC0 (ADC0)	ADC input channel 0.

I/O Ports

The ATMega328P has 23 general purpose digital I/O pins assigned to 3 GPIO ports: 8-bit ports B and D, and 7-bit port C. Each I/O port pin may be configured as an output with symmetrical drive characteristics, each pin has 20 mA of current to drive LED displays directly. It can be configured as an input with or without pull-up resistors (values of 20-50k-Ω). Port B, C, and D, are bi-directional I/O ports. Port B ranges from PB7 to PB0, port C ranges from PC5 to PC0, and port D ranges from PD7 to PD0.

B. *Interrupts*

Interrupt is a signal sent to the CPU by external devices, normally by I/O devices, indicating an event that needs immediate attention. It alerts the processor to a high-priority process requiring interruption of the current process. When an interrupt occurs, the controller completes the current instruction and begins an Interrupt Service Routine (ISR) or Interrupt Handler. When an interrupt happens, ISR instructs the processor or controller what to do.

It is a device that allows modules such as I/O or memory to interrupt the CPU's normal processing. External devices are slower than the CPU in comparison. If there isn't an interrupt, the CPU can spend a lot of time waiting for external devices to catch up to the CPU's speed. The CPU's performance reflects the impact. As a result, an interrupt is needed to remove these limitations.

Interrupt Service Routine (ISR)

An ISR (also known as an interrupts handler) is a software process that is triggered by a hardware interrupt request. It takes care of the request and sends it to the CPU, interrupting the current task. The process is resumed once the ISR is completed. A basic example of an ISR is a routine that handles keyboard events, such as pressing or releasing a key. The ISR processes the input each time a key is pressed.

Kinds of Interrupts

1) Hardware Interrupts are generated by hardware devices to indicate that they need attention from the OS. They may have recently obtained data (for example, keystrokes on the keyboard or data from the ethernet card), or they may have recently accomplished a task that the operating system had previously requested, such as data transfer between the hard drive and memory.

2) Software Interrupts are developed by programs when they want the operating system to conduct a system call on their behalf. An extraordinary state or a special instruction in the instruction set that induces an interrupt when executed by the processor causes the interrupt. For instance, if the processor's arithmetic logic unit executes a command to divide a number by zero, a divide-by-zero exception is triggered. As a result, the machine will either stop calculating or show an error message.

C. *ATMega328p External Interrupts and Pin Change Interrupts*

There are 25 interrupt sources supported by the ATmega328P. Each of these interrupts, as well as the separate Reset Vector, has its own program vector in the Flash program memory space, which is located at the lowest addresses. Table 2 shows the vector table of the two hardware external Interrupts of an ATMega328p.

TABLE II. ATMEGA328P VECTOR TABLE

Source	**Program Address**	**Interrupt Definition**	**ISR() Vector Name**
INT0	0x0002	External Interrupt Request 0 (pin D2)	(INT0_vect)
INT1	0x0004	External Interrupt Request 1 (pin D3)	(INT1_vect)
PCINT0	0x0006	Pin Change Interrupt Request 0 (pins D8 to D13)	(PCINT0_vect)
PCINT1	0x0008	Pin Change Interrupt Request 1 (pins A0 to A5)	(PCINT1_vect)
PCINT2	0x000A	Pin Change Interrupt Request 2 (pins D0 to D7)	(PCINT2_vect)

External Interrupt Registers (EICRn)

ISCx0	**ISCx1**	**Interrupt Definition**
0	0	Low level of INTx generates an interrupt request

0	1	Any logic change on INTx generates an interrupt request
1	0	The falling edge of INTx generates an interrupt request
1	1	The rising edge of INTx generates an interrupt request

External Interrupt Mask Register (EIMSK)

If INTx bit is set (and the SREG I-bit is set), then interrupts are enabled on pin INTx

External Interrupt Flag Register (EIFR)

Interrupt flag bit is set when a change triggers an interrupt request

- Flag is cleared automatically when interrupt routine is executed
- Flag can be cleared by writing a 1 to it

Pin Change Interrupt Control Register (PCICR)

- PCIEn=2 enables interrupts for PCINT[23:16]
- PCIEn=1 enables interrupts for PCINT[14:8]
- PCIEn=0 enables interrupts for PCINT[7:0]

Pin Change Mask Register (PCSMKn)

- PCINTn=2
  controls whether interrupts are enabled PCINT[23:16]
- PCINT=1
  controls whether interrupts are enabled PCINT[14:8]
- PCINT=0
  controls whether interrupts are enabled PCINT[7:0]

III. Circuit Configurations

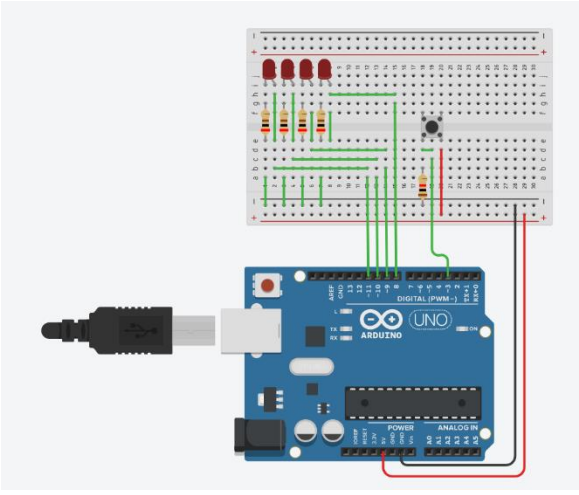

Fig. 2. Using one external interrupt to count up in binary

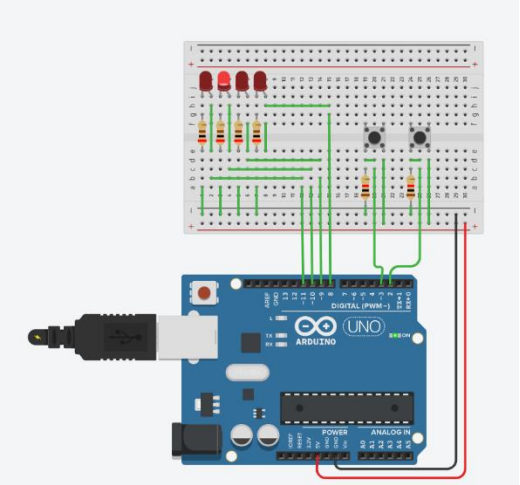

Fig. 3. Using external interrupts to count up in binary and another to reset

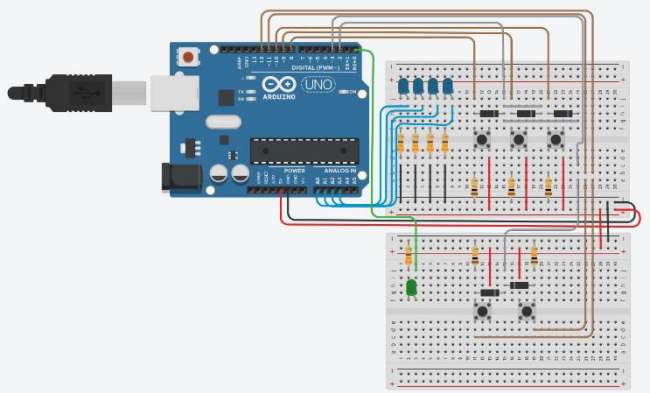

Fig. 4. Using multiple hardware connected to one external interrupt executing different and/or single function/s

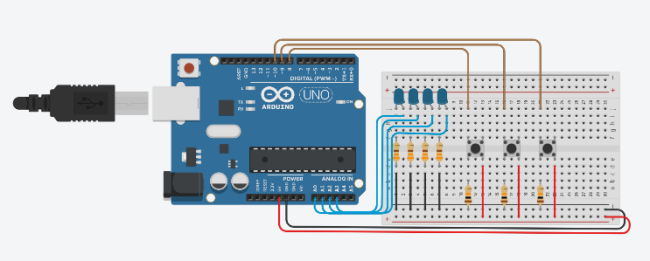

Fig. 5. Using Pin Change Interrupt to Control multiple external interrupts

IV. Programming

A. *Using one external interrupt to count up in binary*

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

```
void setupInterrupt(){

        /* Disable Global Interrupts */
        cli();

        /* Low level of INT1 generates an interrupt
        request */
        EICRA |= (1 << ISC10);

        /* Any logic change on INT1 generates an
        interrupt request */
        EICRA |= (1 << ISC11);

        /* Enable Interrupt 1 */
        EIMSK |= (1 << INT1);

        /* Turn on interrupts */
        sei();

}

int main(void)
{

        setupInterrupt();
        /* Set PORT B as output */
        DDRB = 0xFF;

        /* Set PORT D as input */
        DDRD = 0x00;
        /* Turn off LED initially */
        PORTB = 0x00;

        while(1){}
}


ISR (INT1_vect)
{
        /* Increment PORT B */
        PORTB += 1;
}
```

B. Using external interrupts to count up in binary and another to reset

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


void setupInterrupt(){

        /* Disable Global Interrupts */
        cli();

        /* Low level of INT0 generates an interrupt
        request */
        EICRA |= (1 << ISC00);

        /* Any logic change on INT0 generates an
        interrupt request */
        EICRA |= (1 << ISC01);

        /* Enable Interrupt 0 */
        EIMSK |= (1 << INT0);

        /* Low level of INT1 generates an interrupt
        request */
        EICRA |= (1 << ISC10);

        /* Any logic change on INT1 generates an
        interrupt request */
        EICRA |= (1 << ISC11);

        /* Enable Interrupt 1 */
        EIMSK |= (1 << INT1);

        /* Turn on interrupts */
        sei();

}

int main(void)
{
        setupInterrupt();
        /* Set PORT B as output */
        DDRB = 0xFF;

        /* Set PORT D as input */
        DDRD = 0x00;
        /* Turn off LED initially */
        PORTB = 0x00;

        while(1){}
}

ISR (INT0_vect)
{
        /* Turn off LED when INT0 is triggered */
        PORTB = 0x00;
}


ISR (INT1_vect)
{
        /* Increment PORT B */
        PORTB += 1;
}
```

C. Using multiple hardware connected to one external interrupt executing different and/or single function/s

```
void setupInterrupt(){

 /* Disable Global Interrupts */
        cli();


/* rising edge of INTx generates an interrupt request */
        EICRA |= (1 << ISC00);
        EICRA |= (1 << ISC01);

        EICRA |= (1 << ISC10);
        EICRA |= (1 << ISC11);

 /* Enable Interrupt */
        EIMSK |= (1 << INT0);
        EIMSK |= (1 << INT1);

        /* Turn on interrupt */
        sei();

}


void buttonFunction(){
/* Every button has its specific function */

        if (PINB & 0x01) {
                PORTC += 1;
        }
        if (PINB & 0x02) {
        PORTC -= 1;
        }
        if (PINB & 0x04) {
        PORTC = 0x00;
        }
 }



int main(void)
{

         /* Set PORT B as input and C as output */
         DDRB = 0x00;
         DDRC = 0xFF;
        /*INT0 set as input */
         DDRD = 0b11110011;


        setupInterrupt();

        while(1){}
}


ISR (INT0_vect) {

        buttonFunction();
```

```
}

ISR (INT1_vect)
{
        /* Any button toggles PD0 */
        PORTD ^= (1 << PD0);
}
```

D. *Using Pin Change Interrupt to control multiple external interrupts*

```
void setupInterrupt(){

        /* set PCIE0 to enable PCMSK0 scan */
        PCICR |= (1 << PCIE0);

        /* set PCINTx to trigger an interrupt on state
        change   */
        PCMSK0 |= (1 << PCINT0);
        PCMSK0 |= (1 << PCINT1);
        PCMSK0 |= (1 << PCINT2);

         /* Turn on interrupts */
         sei();

void buttonFunction(){

/* Every button has its specific function */

        if (PINB & 0x01) {
                PORTC += 1;
        }
        if (PINB & 0x02) {
        PORTC -= 1;
        }
        if (PINB & 0x04) {
        PORTC = 0x00;
        }
 }


int main(void)
{

        /* Set PORT B as input and C as output */
        DDRB = 0x00;
        DDRC = 0xFF;
        /*INT0 set as input */
        DDRD = 0b11110011;

        setupInterrupt();

        while(1){}
}


ISR (PCINT0_vect) {

        buttonFunction();
}
```

REFERENCES

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

[1] "EXTERNAL INTERRUPTS ON THE ATmega168/328." Internet: https://sites.google.com/site/qeewiki/books/avr-guide/external-interrupts-on-the-atmega328?fbclid=IwAR1oAhzw4pPMl4eJ_mef_ILE1s0E-V8vUpGPbQugrrLOXkq1LVP4DN4YmNM, [March 25, 2021]

[2] "Global Manipulation of the Interrupt Flag." Internet: https://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html, Feb. 8, 2016 [Mar. 25, 2021].

[3] J. M. Fiore. "29.2: External Interrupts." Internet: https://eng.libretexts.org/Bookshelves/Computer_Science/Book%3A_Embedded_Controllers_Using_C_and_Arduino_(Fiore)/29%3A_Bits_and_Pieces__Interrupts/29.2%3A_External_Interrupts, Sep. 2, 2020 [Mar. 25, 2021].

[4] M. A. Mazidi and S. Naimi. "The AVR Microcontroller and Embedded Systems using Assembly and C." Internet: http://web.csulb.edu/~hill/ee346/Lectures/10%20ATmega32U4%20Interrupts.pdf, Feb. 2009 [Mar. 23, 2021].

[5] M. A. Mazidi and S. Naimi. "#11 ATMEGA328P External Interrupts." Internet: https://www.arxterra.com/11-atmega328p-external-interrupts/, Aug. 22, 2018 [Mar. 25, 2021].

[6] Y. Kardid. "External interrupt on atmega328p." Internet: https://Electronics.Stackexchange.Com/q/324891. https://electronics.stackexchange.com/questions/324891/external-interrupt-on-atmega328p?fbclid=IwAR20OJR_OU4mWBzYnMZ0txxrBlWV7NEGz6kHAzwJ5ZWvLqI9QkdjPWjhf_c, August 20, 2017 [March 25, 2021]