Centria
UNIVERSITY OF APPLIED SCIENCES

Smart Home Project

ICH controller using Arduino

Thibault Mollier

2019

Contents

1 Introduction

The goal of this project is to design and implement a system that uses existing IHC smart home output modules to control lights and some other targets. The main objective is to make it possible to control the output modules without the IHC controller. IHC controller and output modules communicate via serial data transmission. Output modules are composed of 8 channels and the controller shall be able to manage 8 output modules. The output's values will be set by HMI device, the data shall be transmitted to the IHC controller. In our case the controller will be an Arduino uno with Atmega 328P microcontroller [1].

2 Frame

The Arduino shall be able to generate the following frame all the time.

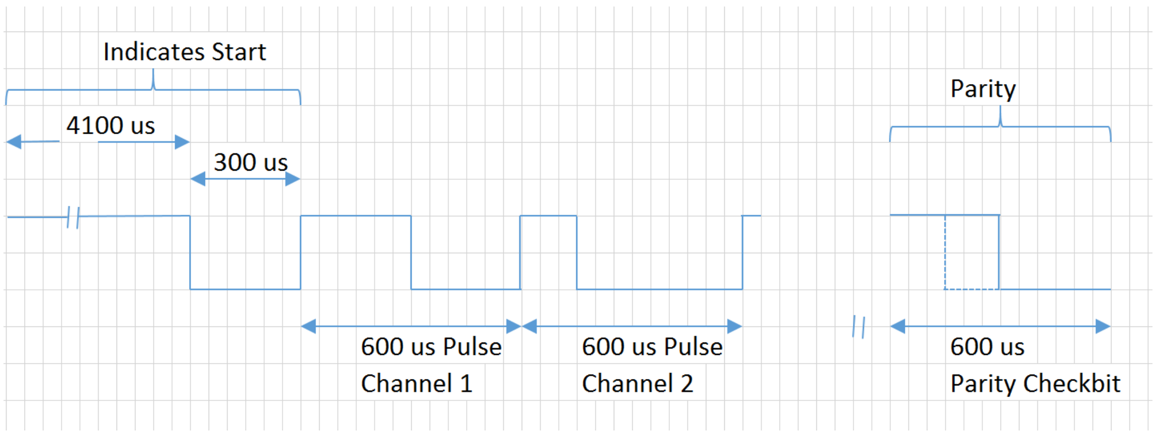


Figure 1: Output Module Control Protocol – voltage levels are between 0 – 5 V

The frame is composed of a start pulse, the data and the parity bit. The data contain 16 bits, but we use only 8 bits (last 8 bit are set to 0). Each bit last 600us and shall start with high state, a duty cycle (high time/total time) of 0.5 is interpreted as a 0 and a duty cycle of 0.25 is interpreted as a 1. The frame shall not be interrupted, consequently our program shall be in real time and use timer interrupt.

2.1 Timer/Counter 1

Timer 1 is a 16-bit timer, it counts from 0 to 0xFFFF. We use 16MHz frequency, that way the timer will overflow after 4.1ms (Start pulse). To generate data bit we use compare interrupt, we compute their values:
$n = t \cdot f$ for t=150us $n = 150 \cdot 16 = 2400$
The counter shall count 2400 for a delay of 150us.

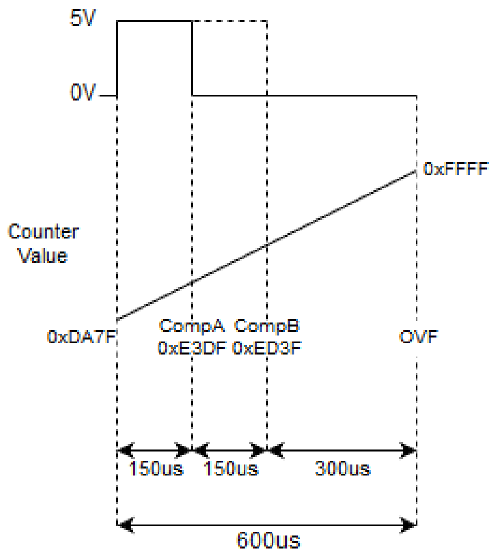


Figure 2: Timer 1 interrupt

On overflow we set the output to 1, on compA interrupt we set the reverse value of data then we set the output to 0 on compB.

We need to allow interrupt and set the timer in CTC mode, we obtain the following code to initialize the timer :

```
cli();//stop interrupts

//set timer1 interrupt
TCCR1A = 0;// set entire TCCR1A register to 0
TCCR1B = 0;// same for TCCR1B
TCNT1  = 0;//initialize counter value to 873

// turn on CTC mode
TCCR1B |= (1 << WGM12) | (1 << WGM13);

// no prescaler
TCCR1B |= (1 << CS10);

// set compare match register
OCR1A = 58335;
OCR1B = 60735;

// enable timer compare and overflow interrupt
TIMSK1 |= (1 << TOIE1) | (1 << OCIE1B) | (1 << OCIE1A);

sei();//allow interrupts
```

Now we know how to generate each part of the frame, but we shall order it with a state machine.

2.2 State machine

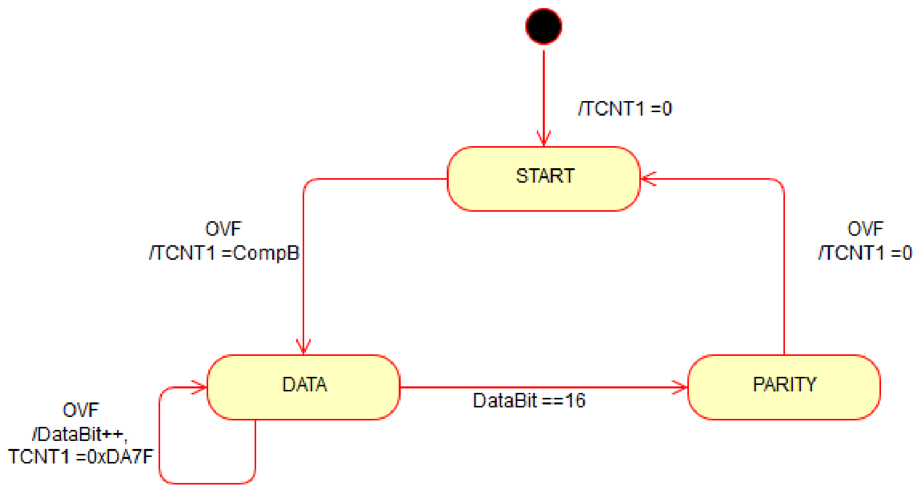


Figure 3: State machine

First, we generate the 4.1ms pulse by counting from 0 to the maximum value (0xFFFF), when the counter overflow we jump to the DATA state and we set the value of TCNT1 register to 0xED3F, that way the next overflow will be in 300us, then we can start to generate data as previously explained. When we transferred all bit, we write the parity bit and then we return to the start state and go on.

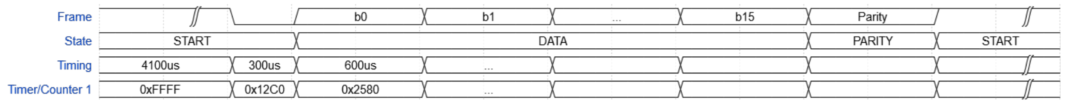


Figure 4:

```
/*
        Interrupt on CompA register
*/
ISR(TIMER1_COMPA_vect){
        switch (FRAME)
        {
                case START:
                        break;
                case DATA:
                        if (Hwapi.AllDataTransmitted())
                        {
                                FRAME=PARITY;
                                Hwapi.WriteParity();
                        }else{
                                Hwapi.Write(DataIn);
                        }
                        break;
                case PARITY:
                        break;
        }
}
/*
        Interrupt on CompB register
*/
ISR(TIMER1_COMPB_vect){
        switch (FRAME)
        {
                case START:
                        break;
                case DATA:
                case PARITY:
                        Hwapi.Write(LOW);
                        break;
        }
}
```

```
/*
        Interrupt on overflow
*/
ISR(TIMER1_OVF_vect){
        switch (FRAME)
        {
                case START:
                        Hwapi.Write(LOW);
                        Timer1.time_to_OVF(t_300us);
                        FRAME=DATA;
                        break;
                case DATA:
                        Hwapi.Write(HIGH);
                        Timer1.time_to_OVF(t_600us);
                        break;
                case PARITY:
                        Hwapi.Write(HIGH);
                        Timer1.time_to_OVF(t_4100us);
                        Transmiter.read(DataIn);
                        FRAME=START;
                        break;
        }
}
```

3 Serial

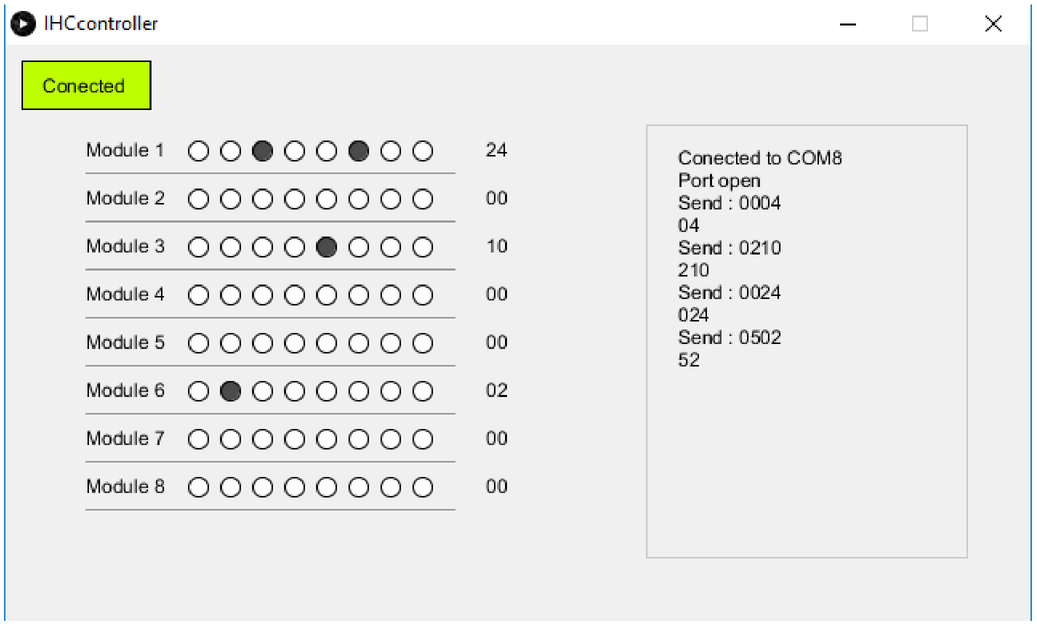
We have 4.1ms unused during the start pulse, we can take advantage of this time to read and write the serial.

3.1 Protocol

Firstly, we shall wait the "port open" write at the end of the initialization, afterward we can send command. Command are composed of two bytes: The first one indicates the module number (number between 0 and 7), the second indicate the configuration of this module.
Example: 0x05FF set all module 5 output to one.
Then the program confirms the change by replying the new configuration. In case of no reply after 10ms you should resend the command. There is a 64 bytes buffer in the Atmega328 it should be possible to send several commands successively (ex: 0x04AC05FF0112) but the program process it by pair of bytes.

4 HMI example

This is an example of a software able to control the IHC controller from a computer. It is writen in java using Processing, the source code is available.



```
Conected to COM8
Port open
Send : 0004
04
Send : 0210
210
Send : 0024
024
Send : 0502
52
```

Figure 5: IHM

References

[1] Microchip. *megaAVR® Data Sheet*. 2018.
http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf