

R/C pulse output unit based upon a 74HC4017 decade counter.

Refer to the timing diagram, below, and note, reset drives all output low EXCEPT Q0, which is driven high. The general algorithm is to clock the unit the pulse width value after an output goes high. So, after reset, enable the timer subsystem to give a positive pulse after X microseconds. After the clock, Q1 will be high and Q0 will be low, so reset the timer system to give the next pulse at the proper delay for Q1, and so forth. I happened to pick port B2 as the reset line, but any unused line would do.

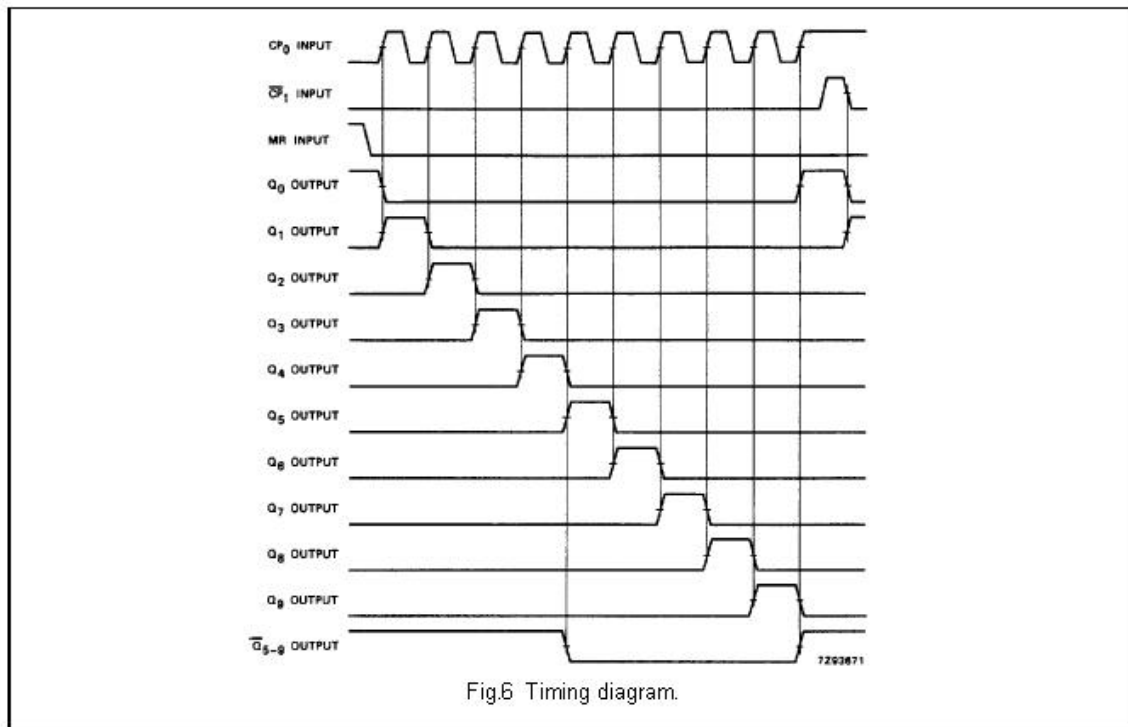
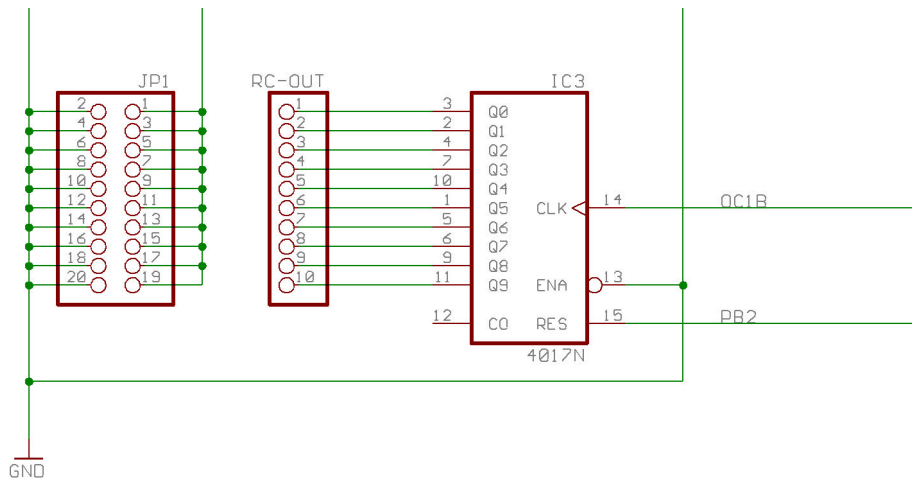


Fig.6 Timing diagram.

On the AVR I used timer1 (16bit) that has two output compare units (OC1A and OC1B). I set the timer system to run at clock/8, which is .5us on a 16 MHz system, and “set on compare match” mode. I force the output low, set the compare value based upon the current timer value + the delay, then enable “set on match” and return. Upon match, OC1B goes high, an interrupt is generated and the process continues. When the 9th value has been generated the interrupt handler disables the interrupt and no more pulses are generated. At a higher level of tasking (e.g. my main motor loop) I call the start routine to output the 9 values. That occurs every 24ms in my system.

This would be easy to extend to 18 outputs by adding a second decade counter and driving it off OC1A. A second interrupt handler would be needed and the start routine would need to be modified to start both timer compare handlers.

A side note: the code was written to be as small as possible. The start routine assumes the interrupt is present, so it just sets the compare unit to the timer value and enables the interrupt. The interrupt handler then adjusts the compare for the proper value, etc. The very first time after reset there is no interrupt flag (nothing has happened yet) so the timer has to wrap to generate the first interrupt. This takes 32ms on a 16mhz system. Not a big thing.

The following code sample is for GCC, the free Gnu C/C++ compiler for the AVR processor. This code should work unmodified on any AVR chip that has the 16 bit timer1.

```

// HC4017 Johnson Decade Counter R/C pulse decoder.
// Array RCpulse contains the pulse width, in .5us (1us)
// Call StartRCout() to initialize the process. The interrupt
// Handler will terminate after issuing 9 pulses.

uint16_t RCpulse[9], *iRC;

void InitRCout(void)
{
    unsigned char i;
    //
    // High Speed Counter settings: CLK/8 normal .5us resolution
    //
    TCCR1A = ((1<<COM1A1) | (1<<COM1B1));
    TCCR1B = (1<<CS11);
    //
    for (i=0; i<9; i++)
        RCpulse[i] = 3000; // Put everyone at neutral (1.5 ms).
}

void StartRCout(void)
{
    unsigned char tmp = SREG;

    cli();
    PORTB |= BV(RC_Reset); // Reset counter (set out0 high)
    OCR1B = TCNT1; // Capture our start time reference
    PORTB &= (char)~BV(RC_Reset);
    SREG = tmp;
    iRC = &RCpulse[0]; // Point to first entry
    TIMSK |= BV(OCIE1B); // Enable Compare on match interrupt
}

SIGNAL(SIG_OUTPUT_COMPARE1B)
{
    TCCR1A &= (char)~_BV(COM1B0); // Clear on match
    TCCR1A |= _BV(FOC1B); // Force
    OCR1B += *iRC++; // Calculate time to next compare.
    TCCR1A |= _BV(COM1B0); // Set on match
    if (iRC > &RCpulse[8]) // Disable interrupts on completion
    {
        TIMSK &= (char)~BV(OCIE1B);
    }
}

```