Tiny Microcontroller Implements a Gated Test Pattern Signal Generator

Dhananjay V. Gadre
dvg@iucaa.ernet.in
Inter-University Centre for Astronomy and Astrophysics
Pune - 411007, INDIA

It was desired to have a gate signal controlled clock generator. While on the face of it, it seemed that a simple AND gate could be used to gate a clock signal. However, what was also desired was to have a control over the clock frequency as well as multiple test signals. These test signals were required for testing the data input performance of a PCI based data acquisition system.

An implementation based on a PAL device (GAL22V10) was found to be too restrictive and so we decided to consider the AVR series of RISC controllers to implement our signal generator.

Figure 1 illustrates our design. The circuit uses an 8-pin AVR controller, the AT90S2343. The AVR controllers are In-system programmable (ISP) which is an advantage when you want to make modifications fast and often and without having to remove the controller out of the target.

In its simplest implementation, the AT90S2343 controller was programmed to generate a gated clock signal which was then used to trigger the PCI based data acquisition system.

The gate signal is applied to an external interrupt pin (INT0) of the AT90S2343 processor. The interrupt pin is programmed such that at startup, the Interrupt Subroutine (ISR) is executed at the rising edge of the gate pulse and subsequently, the ISR toggles the edge sensitivity of the INT0 interrupt pin. The next interrupt is generated on the falling edge of the gate pulse and so on.

The ISR setups a flag register which determines whether the test pattern generator routine is to be executed or not. When the flag is set to FF(hex), the test pattern generator is executed. The flag is set to FF(hex) at each rising edge of the gate pulse and is reset to 00 on the subsequent falling edge of the gate pulse.

The ISR itself is implemented in an unusual manner. Typically, the ISR always ends with a Return from Interrupt instruction (e.g. the RETI instruction for the AVR controllers). However, this is not necessary as long as the program takes care that the stack pointer does not run away and the interrupts are enabled again. In our program, it was not required that the ISR return to the interrupted program after the completion of the ISR. At the end of the ISR, the program returns to a fixed location in the main program where the stack pointer is readjusted, the interrupts are enabled and the program waits for the next occurrence of the interrupt. If the flag is set up by the last ISR such that the test pattern generator is to be executed, then the program execution proceeds to the test pattern generator subroutine, otherwise it just waits in a loop for the flag value to be toggled again by the ISR.

The test pattern generator can be modified to generate as many as three test signals, by modifying the gate_clk subroutine; the frequency of the gated signals can be adjusted by adding delay. The subroutine can be implemented so as to generate any complex waveform sequence.

Figur 2 illustrates once simple case where a gated clock is generated by the circuit.

```
;gclk.asm
;Dhananjay V. Gadre
;dvg@iucaa.ernet.in
;September 1, 2000.
;Program to generate an arbitrary test pattern controlled by a
;gate signal.

.include "2323def.inc"

.def temp=r17
```

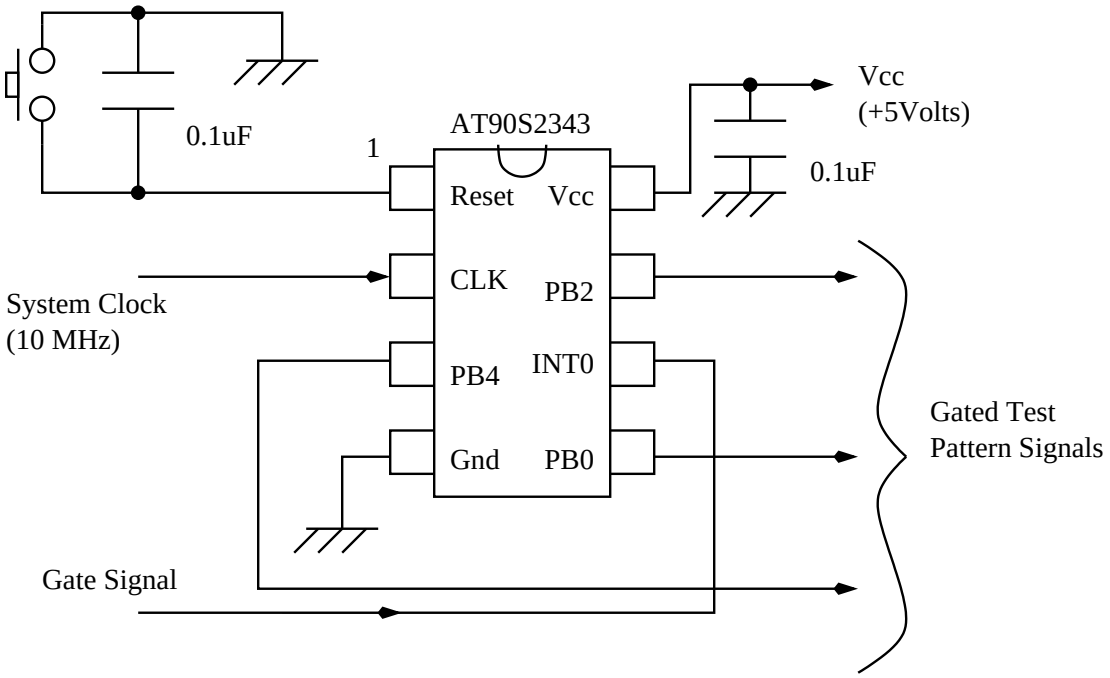

Figure 1: Gated Test Pattern Signal Generator

```
.def t_flag=r21
.def temp1=r22
.def stackload=r18

.cseg
.org 0
        rjmp    RESET           ;Reset Handle
        rjmp    ex0
        rjmp    RESET

RESET:
       ldi temp, 0b00011101 ;configure PORT B
       out DDRB, temp       ;PB1 as input rest all outputs.

       ldi temp, 2   ; PB1 is set to 1 for enabling pullup
       out PORTB, temp

       ldi t_flag, 0

       ldi temp, $40     ;enable INT0
       out GIMSK, temp

       in temp, mcucr   ;set INT0 interrupt for rising edge
       ori temp, $03
       out mcucr, temp

       ldi temp1, 1 ;register used to toggle INT0
                    ;edge sensitivity.
```

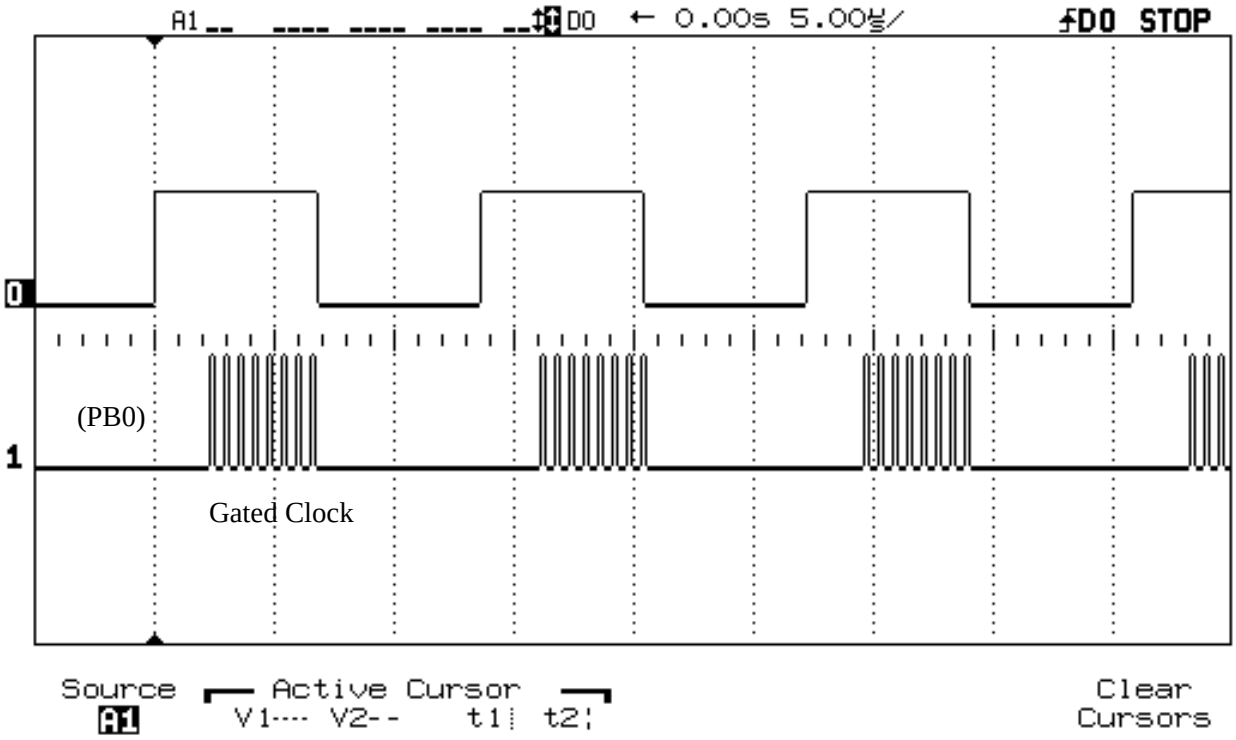

Figure 2: Gated Test Clock Signal Output

```
       ldi stackload, low(RAMEND)
ret_from_ints:
       out SPL, stackload    ;init the stack pointer.
       sei                   ;enable global interrupts


start:
       cpi t_flag, 0         ;wait till the flag is set
       breq start

       rjmp gate_clock      ;flag is set, go generate
                     ;test signals.
       rjmp start

;+++++++++++++++++++++++++++++++++++++++++++++++
;routine geretaes the test signals.
;Currently, only generates a clock signal on PB0.
;but can be modified to generate more elaborate
;test signals. 2 more output lines:  PB2 and PB4
;are available.
;+++++++++++++++++++++++++++++++++++++++++++++++
```

```
gate_clock:
       sbi PORTB, 0
       cbi PORTB, 0
       rjmp gate_clk


;+++++++++++++++++++++++++++++++++++++++++++++++
;INT0 ISR.
;INT0 is generated at the rising as well as the
;falling edge of the gate signal on pin PB1. At startup
;the interrupt is setup to occur on the rising edge.
;The ISR toggles the edge sensitivity of INT0 from
;a rising edge to falling edge or falling edge to rising
;edge at each interrupt occurance. The ISR also toggles the
;value of t_flag which is used to start or end the
;test pattern generation.
;+++++++++++++++++++++++++++++++++++++++++++++++
ex0:   cbi PORTB, 0
       com t_flag
       in temp, mcucr
       eor temp, temp1
       out mcucr, temp ; toggle the INT0 edge

;Note the unusal return from an interrupt subroutine.
;usually, the ISR returns back with a RETI (return from Interrupt)
;instruction. However, we return to a known point in the main
;program where the stack pointer is readjusted and the interrupts
;are again enabled, something that the RETI instruction does.

       rjmp ret_from_ints
```