Auto-Adjustment of USB Monitor Screen Brightness

Introduction:

Non-critical measurement of resistance has been widely applied in many measurement fields such as temperature, light, etc., measurement where the resistance is the sensor. In such applications, one Microcontroller(*mP*) is often required to control the charging/discharging of capacitor and counting the time constant of this charging process. That will reduce the measurement cost with no real ADC chip required. The automatic adjustment of USB monitor screen brightness is exactly such an example. On one hand, update of USB monitor brightness is not highly accurately required, on the other hand, USB can be used to transfer the brightness between monitor and PC, that provides one flexible way to modify monitor characteristic. In this application note, one typical circuit based on Philips *mP* , 87C51, in USB scenario was proposed, firmware to implement the equivalent A/D conversion was also given with assembly code.

Fundamental and hardware:

Following figure describes one typical hardware implementation. One open drain (OD) IO pin of *mP* was used to control charging/discharging process. The capacitor voltage is used as $\overline{INT0}$ which can be set as the external control of internal counter operation. Two factors feature hardware design regarding to USB requirements.

- Firstly, USB traffic should have the highest priority to use CPU bandwidth if a *mP* involves in USB implementation. That requires *mP* take as less time as possibly to service other things, and be not allowed to turn to other actions during USB transaction services. Other routines have to wait forever till *mP* finish USB transaction service. But the charging capacitor is a real-time process, so the timing of such charging process must be hardware-controlled to stop.
- If non-OD port pin was chosen, VCC will also charges the capacitor through the internal pullup resistance of *mP* . That requires extra calibration and add not required time cost.

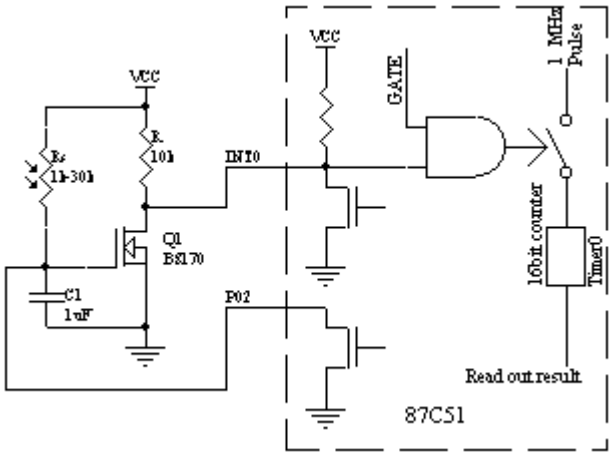

Figure 1 Diagram of ambient brightness measurement through RC circuit

Auto-Adjustment of USB Monitor Screen Brightness

As shown in figure, P0.2 is used to control the switching of charging and discharging capacitor C1, $\overline{INT0}$ is set as the external control for internal counter operation. This signal is low active, means the counting was forced to stop once this signal becomes low. Q1 is a general TTL-compatible switch transistor that is used for logic conversion. When P0.2 becomes low, capacitor discharges through internal port resistor fast, $\overline{INT0}$ becomes high. After P0.2 switches high by firmware, VCC charges the capacitor through Light Dependent Resistor (LDR), in the meantime, as long as the GATE was set as 1, the internal counter was enabled only by $\overline{INT0}$. Upon the voltage of P0.2 rising up to the threshold at which the transistor turns on, $\overline{INT0}$ becomes low, counting stop. The time constant of charging remains in timer register. Once USB transaction was finished, mP reads the data and stores it to one specific memory in the form of USB report, and begins next measurement procedure in a short time.

One point must be noted, CAP must be discharged sufficiently before counting the charging time, this is disadvantageous for USB transaction. Fortunately, the CAP discharges through the internal MOS gate of port 0 in above circuit, the open resistance of this GATE is very low, so discharging time is very short.

Assuming that V_T is the threshold where the inverter outputs low level, the time of capacitor voltage rising up to this threshold is

$$t = -R_s C \ln\left(1 - \frac{V_T}{VCC}\right)$$

here R_S is the resistance of the LDR, C is the capacitance of the CAP. Under constant C and VCC, the time is proportional to the resistance of the LDR. As R_S corresponds solely to the ambient brightness variation, by counting the charging time of the capacitor, the ambient brightness can be measured. USB monitor driving software should be able to get this measurement report, and adjust the brightness of monitor screen by it. Next figure shows the CAP charging/discharging waveform:

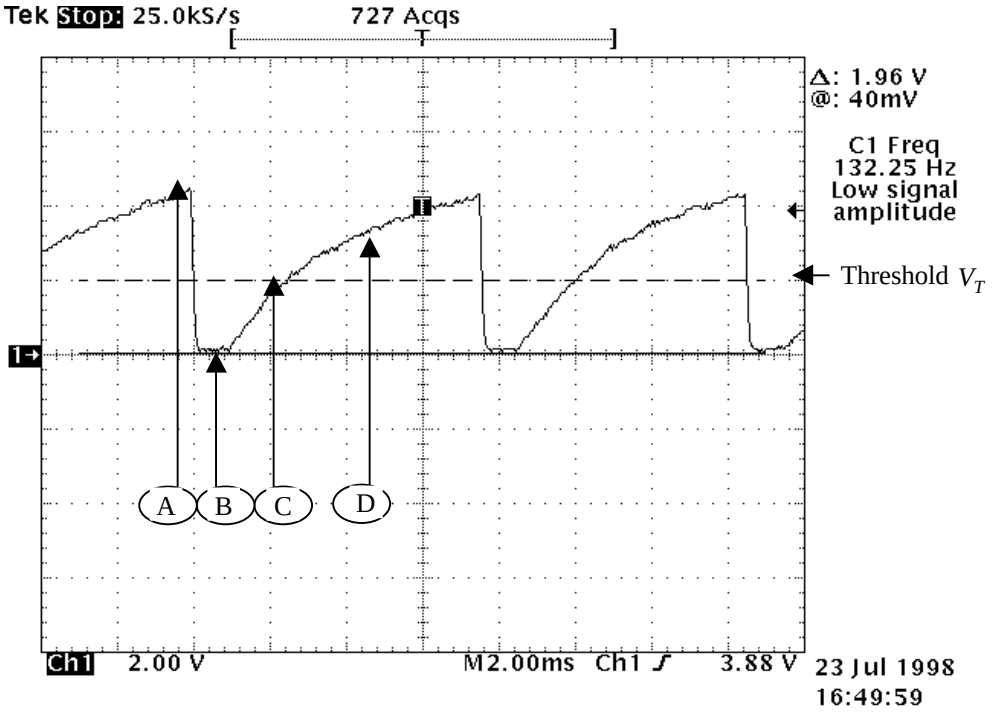

Figure 2 Waveforms of CAP voltage with USB transaction and without USB transaction

The meanings at every point in figure are following:

A. Discharges CAP;
B. Release CAP, counting starts;
C. Cross threshold, leads to counting stop;
D. "Dead" time due to *mC* busy processing other tasks.

From this figure we know, during the Microcontroller services USB transaction, CAP will be charged forever until the USB transaction was finished. As shown in figure, the threshold V_T is about 2.0V. Delay time begins from discharging point, in this implementation, we set delay time as only 64 *ms* :

```
      MOV   08H,#20H    ;delay 20*2 us to discharge the capacitor
HERE2:
      DJNZ  08H,HERE2   ;delay to ensure CAP be fully discharged
```

because the discharging time is very fast. Measured time is about 1720ms

Implementation and firmware:

Before realizing auto-adjustment of monitor contrast, several issues must be considered:

- Measurement error. Due to the temperature effect and instability of light flux through the LDR, the resistance of this sensor is unstable, that leads to large measurement error. Relative measurement error, or accuracy, of the LDR is about ±17%, which is compliant to temperature effect value given by LDR specification. Accuracy is one factor that determines the minimum wordlength of counter. Another error source is the quantization error that depends on the counter operation frequency. If the OSC frequency is 12MHz, the counting frequency of timer0 is 1MHz, the minimum counting error is 1μs. This quantization error requires the minimum timing constant should be greater than 1μs. That requires a larger capacitor be used. For example, in figure.1, one CAP of 1μs was used.
- Dynamic range. Assuming that 40dB is enough for the ambient brightness variation of a monitor. Then LDR resistance may varies from 500Ω to 50kΩ for the selected LDR corresponding to from the brightest to the darkest. The dynamic range of 40 dB requires the effective wordlength of 7bits. To tolerate larger variation of threshold and reduce the effect of quantization, 16bit counter was chosen.
- Calibration of brightness coefficient. There are two types of calibration issues. The first is to scale the coefficient assuming the linear relationship between resistance of LDR and ambient brightness. In fact, the actual value of the LDR resistance doesn't matter, what we need is the ambient brightness coefficient respect to the LDR resistance. This value is usually not available. From above equation, we know the measurement depends on also the capacitor value and how large threshold voltage is. For a peculiar design, we can estimate the timing constant from specifications given by component datasheets. However, this manually calibration is only a reference for component selection and because there is much difference between devices, in-line calibration for such measurement system is necessary. The second issue of calibration is whether the relationship between sensor and ambient light strength is linear or not. If not, one additional calibration such as curve fitting is needed. However, this data processing cannot be finished in *mP* because it has only limited memories. But it can be done in Host after PC receives the data through USB. Currently, we assume the linear relationship and only consider the first calibration issue. Calibration must be finished before true measurement. We have done this task during component selection.

In firmware, interrupt from $\overline{INT0}$ must be disabled and masked. Timer0 must be defined as one16-bit counter, relevant configuration was set as following:

```
                    .
                    .
                    .

DISCHARGE           BIT   P0.2        ;low,discharge CAP;high, counting command
GATINGPIN           BIT   P3.2        ;gating counting, high, counting, low, stop
BRIGHTNESS          DATA  26H         ;brightness data buffer
```

Auto-Adjustment of USB Monitor Screen Brightness

```
                .
                .
                .
;
;  initialize the ADC hardware and registers
;
      CLR   EA          ;disable all interrupts
      CLR   ES          ;disable each individual interrupt
      CLR   ET1
      CLR   EX1
      CLR   ET0
      CLR   EX0
;
;     Initialize ADC mechanism with RC
;
      MOV   TMOD,#09h   ;setup timer0 as a 16-bit counter
      SETB  GATINGPIN   ;make P3.2/INT0 as input. Up to now, the CAP should be
                        ;fully charged upon power on, P0.2 is high, P3.2/INT0
                        ;should be low
      CLR   TR0         ;forbid counting
      CLR   DISCHARGE   ;discharge CAP, when pin P0.2 becomes low, P3.2/INT0
                        ;becomes high, allow counting from 0.
      MOV   TH0,#0      ;now P0.2=0, P3.2=1. clear counter value of timer0
      MOV   TL0,#0
      CLR   TF0         ;clear counter of timer0 overflow flag
      SETB  DISCHARGE   ;charge CAP. When CAP voltage is
                        ;greater than one certain value(1.2V in case of HCT04),
                        ;P3.2/INT0 logic converse, counting stops. RC time
                        ;constant was counted and remain in 16-bit counter. But
                        ;the charging continues until uP clears P0.2 to
                        ;discharge.
      SETB  TR0         ;start counting because right now, P3.2/INT0 is high, so
                        ;this instruction begin hardware counting
;------------------
                .
                .
                .
```

When MCU finishes current USB service, it turns to checking if one time of ADC task has completed. If one ADC finishes, it will read out the result from the Timer0 register, and send to Host through interrupt endpoint of embedded function together with three kinds of hotkeys Brightness Up, Brightness Down and Brightness Hotkey:

```
;-------------------------------------------------------------------------
;     ADC process with RC
;-------------------------------------------------------------------------
;
A2D:
      JNB   GATINGPIN,NOT_COUNTING  ;High P3.2 shows CAP is being charged, not
                                    ;arrive the threshold(1.2V)
      JB    TF0,OVFLOW1       ;timer0 is overflow, should stop and restart
                              ;restart counting
      AJMP  A2D_RTRN          ;in charging before arriving threshold
```

Auto-Adjustment of USB Monitor Screen Brightness

```
NOT_COUNTING:
      JB    TF0,OVFLOW        ;totally timer0 is overflow, restart
      MOV   A,TH0
      CPL   A
      AJMP  StoreBrightness
OVFLOW1:
      CLR   TR0         ; stop counting
OVFLOW:
      MOV   A,#0        ; overflow shows in the darkest ambient light
StoreBrightness:
      MOV   BRIGHTNESS,A      ; contrast data into Microprocessor;
;     AJMP  RESTART     ;reinitialize ADC mechanism
;RESTART:               ;reinitialize the ADC mechanism
      CLR   TR0         ;stop counting
      CLR   DISCHARGE   ;discharge CAP,
      MOV   08H,#20H    ;delay 20*2 us to discharge the capacitor
HERE2:
      DJNZ  08H,HERE2   ;delay to ensure CAP be fully discharged
      MOV   TH0,#0      ;clear counter current content of timer0
      MOV   TL0,#0
      CLR   TF0         ;software clear counter of timer0 overflow flag
      SETB  DISCHARGE   ;and begin charging CAP
      SETB  TR0   ;re_ensure timer0 work be controlled only by external
                  ;P3.2/INT0 pin, because right now, P3.2/INT0 is high, so
                  ;this instruction begin hardware counting again
A2D_RTRN:
      RET
                  .
                  .
                  .
```

When Host polls embedded function interrupt EP, Interrupt endpoint will send hotkey-value and brightness value:

```
INT_SEND_TO_HOST:
      MOV   RW_COUNTER,#04                ;TOTAL BYTES TO SEND
      MOV   I2C_BUFFER,#WRITE_BUFFER      ;COMMAND
      MOV   I2C_BUFFER+1,#0         ;FIRST BYTE SHOULD BE ZERO
      MOV   I2C_BUFFER+2,#4         ;DATA LENGTH
      MOV   I2C_BUFFER+3,#02        ;REPORT ID
      MOV   I2C_BUFFER+4,R1         ;key_value. Host driver will act on this
                                    ; value
      MOV   I2C_BUFFER+5,#10        ;REPORT ID(10, brightness)
      MOV   I2C_BUFFER+6,BRIGHTNESS ;brightness data
;--------------
```

Discussion:

There are many ways to fulfill calibration. Selection of calibration depends on detailed application. During development, the logic inverter required above is key component to reduce the cost further. In this application note, one N MOSFET transistor was finally determined. Users can choose any other inverters, such as NOT GATE on board, etc., but whatever inverter was selected, positive input is required.