EE 354 **Fall 2016**
Pulse Width Modulation Notes

Many embedded systems are built to have PWM as a hardware function since PWM is a cheap way to do digital to analog conversion. In PWM we start with a base frequency of say 1,000Hz and each cycle we output a single pulse whose width varies from 0% of the period to 100% of the period.

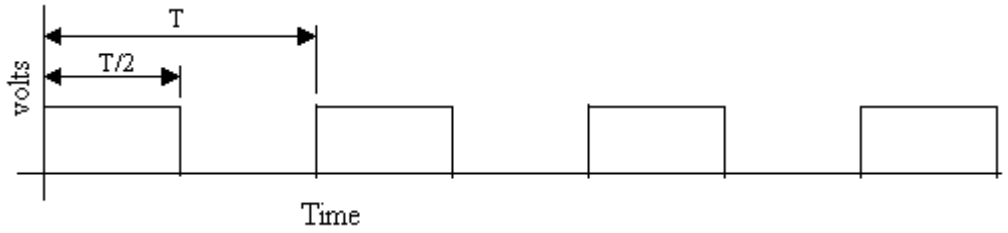

Figure 1
Pulse width modulation. In this case the signal is 50% of maximum.

The base frequency of the PWM is f = 1/T so that the frequency content of a PWM signal will have frequencies at f and higher due to the base frequency. The signal information is transmitted by the duty cycle and will be, generally, a much lower frequency signal. Thus a relatively simple low pass filter can eliminate the frequencies at the base frequency and above and recover an analog signal from the PWM.

The AT89C51CC03 has a Programmable Counter Array (PCA) which support pulse width modulation. The PCA has its own 16-bit timer/counter plus five 16-bit capture registers and the five registers can be used in the following modes:

- 16-bit capture that is positive edge triggered.
- 16-bit capture that is negative edge triggered.
- 16-bit capture that is positive and negative edge triggered.
- 16-bit software timer
- 16-bit high speed output
- 8-bit pulse width modulation

Capture register 4 can also be used as a watchdog timer.

There is only one 16-bit PCA timer and this forms the basis for all of the capture registers. In capture mode, each capture register has an associated external bit. When that bit is triggered, the capture register "grabs" the PCA timer. This capture mode can thus be used to time external signals. The software timer mode is an extension of the use of Timer 0 and Timer 1. In this mode you simply set a value in the capture register and when the PCA Timer reaches that value an interrupt is generated. In the high speed output mode, an external bit can be toggled each time a capture register matches the PCA timer.

For this class, we will be using only the PCA's pulse width modulation mode.

In PWM mode, the base frequency is generated by the PCA timer. The system allows up to five PWM signals to be generated independently of one another except for the base frequency which must be common to all five PWM signals.

Figure 1 shows a block diagram of the PWM operation for the AT89C51CC03.

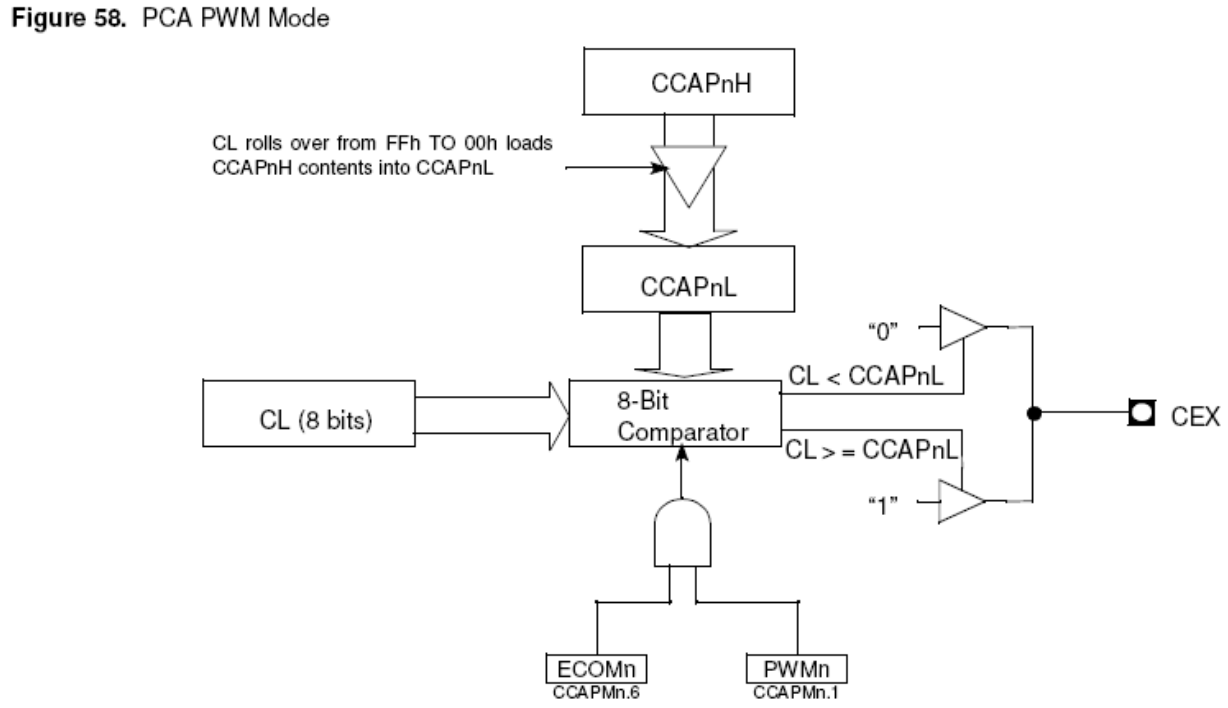

Figure 1
The 8-bit PWM system on the AT89C51CC03 processor.

Note that the PWM system is an 8-bit system and the 16-bit capture registers are divided into a lower part and an upper part which is similar to mode 2 on the timers which does auto-reload.

To do PWM on the AT89C51CC03 you must first set up the PCA timer. This set up is similar to the set up for timer 0 and timer 1. With this set up complete, the PCA timer runs with no further software interaction. Your software then loads the signal value into the high byte of the capture register. This high byte is automatically loaded into the low byte. When the PCA timer value is less than the value in the capture register the PWM bit is set to low. When the PCA timer value is greater than or equal to the capture register value the PWM bit is set to high. After 256 counts, on the PCA counter (CL in Figure 1), the high byte of the capture registers is automatically reloaded into the low byte.

The PCA timer has several sources and may be driven the CPU clock/6, the CPU clock/2, the Timer 0 overflow bit, or from an external source via P1.2. The PCA timer source is determined by the two bits CPS1 and CPS0 in the PCA counter mode register CMOD. The CMOD register is shown in Figure 2 and a block diagram of the PCA counter is shown in Figure 3. Figure 4 shows the Counter control register and Figure 5 shows the capture mode registers.

Table 50. CMOD Register

CMOD (S:D9h)
PCA Counter Mode Register

7	6	5	4	3	2	1	0
CIDL	WDTE	-	-	-	CPS1	CPS0	ECF

Bit Number	Bit Mnemonic	Description
7	CIDL	**PCA Counter Idle Control bit** Clear to let the PCA run during Idle mode. Set to stop the PCA when Idle mode is invoked.
6	WDTE	**WatchDog Timer Enable** Clear to disable WatchDog Timer function on PCA Module 4, Set to enable it.
5	-	**Reserved** The value read from this bit is indeterminate. Do not set this bit.
4	-	**Reserved** The value read from this bit is indeterminate. Do not set this bit.
3	-	**Reserved** The value read from this bit is indeterminate. Do not set this bit.
2-1	CPS1:0	**EWC Count Pulse Select bits** CPS1 CPS0 Clock source 0 0 Internal Clock, FPca/6 0 1 Internal Clock, FPca/2 1 0 Timer 0 overflow 1 1 External clock at ECI/P1.2 pin (Max. Rate = FPca/4)
0	ECF	**Enable PCA Counter Overflow Interrupt bit** Clear to disable CF bit in CCON register to generate an interrupt. Set to enable CF bit in CCON register to generate an interrupt.

Reset Value = 00XX X000b

Figure 2

Counter mode register CMOD for the PCA. Bits 2 and 1 determine the source of the PCA timer.

Figure 53. PCA Timer/Counter

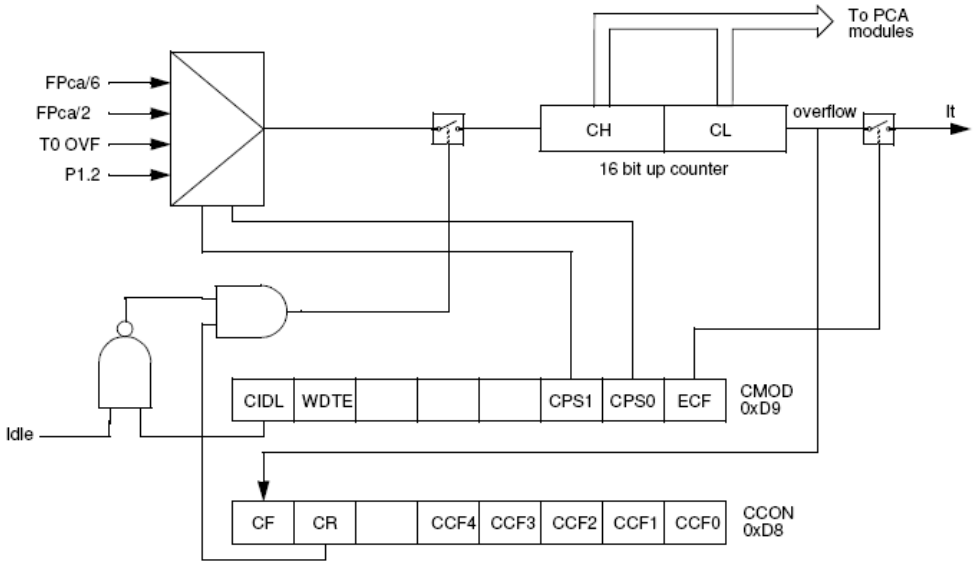

Figure 3

The PCA timer/counter and its control bits.

Table 51. CCON Register

CCON (S:D8h)
PCA Counter Control Register

7	6	5	4	3	2	1	0
CF	CR	-	CCF4	CCF3	CCF2	CCF1	CCF0

Bit Number	Bit Mnemonic	Description
7	CF	**PCA Timer/Counter Overflow flag** Set by hardware when the PCA Timer/Counter rolls over. This generates a PCA interrupt request if the ECF bit in CMOD register is set. Must be cleared by software.
6	CR	**PCA Timer/Counter Run Control bit** Clear to turn the PCA Timer/Counter off. Set to turn the PCA Timer/Counter on.
5	-	**Reserved** The value read from this bit is indeterminate. Do not set this bit.
4	CCF4	**PCA Module 4 Compare/Capture flag** Set by hardware when a match or capture occurs. This generates a PCA interrupt request if the ECCF 4 bit in CCAPM 4 register is set. Must be cleared by software.
3	CCF3	**PCA Module 3 Compare/Capture flag** Set by hardware when a match or capture occurs. This generates a PCA interrupt request if the ECCF 3 bit in CCAPM 3 register is set. Must be cleared by software.
2	CCF2	**PCA Module 2 Compare/Capture flag** Set by hardware when a match or capture occurs. This generates a PCA interrupt request if the ECCF 2 bit in CCAPM 2 register is set. Must be cleared by software.
1	CCF1	**PCA Module 1 Compare/Capture flag** Set by hardware when a match or capture occurs. This generates a PCA interrupt request if the ECCF 1 bit in CCAPM 1 register is set. Must be cleared by software.
0	CCF0	**PCA Module 0 Compare/Capture flag** Set by hardware when a match or capture occurs. This generates a PCA interrupt request if the ECCF 0 bit in CCAPM 0 register is set. Must be cleared by software.

Reset Value = 00X0 0000b

Figure 4

Counter Control register CCON for the PCA.

Table 54. CCAPMn Registers

CCAPM0 (S:DAh)
CCAPM1 (S:DBh)
CCAPM2 (S:DCh)
CCAPM3 (S:DDh)
CCAPM4 (S:DEh)
PCA Compare/Capture Module n Mode registers (n=0..4)

7	6	5	4	3	2	1	0
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn

Bit Number	Bit Mnemonic	Description
7	-	**Reserved** The Value read from this bit is indeterminate. Do not set this bit.
6	ECOMn	**Enable Compare Mode Module x bit** Clear to disable the Compare function. Set to enable the Compare function. The Compare function is used to implement the software Timer, the high-speed output, the Pulse Width Modulator (PWM) and the WatchDog Timer (WDT).
5	CAPPn	**Capture Mode (Positive) Module x bit** Clear to disable the Capture function triggered by a positive edge on CEXx pin. Set to enable the Capture function triggered by a positive edge on CEXx pin
4	CAPNn	**Capture Mode (Negative) Module x bit** Clear to disable the Capture function triggered by a negative edge on CEXx pin. Set to enable the Capture function triggered by a negative edge on CEXx pin.
3	MATn	**Match Module x bit** Set when a match of the PCA Counter with the Compare/Capture register sets CCFx bit in CCON register, flagging an interrupt.
2	TOGn	**Toggle Module x bit** The toggle mode is configured by setting ECOMx, MATx and TOGx bits. Set when a match of the PCA Counter with the Compare/Capture register toggles the CEXx pin.
1	PWMn	**Pulse Width Modulation Module x Mode bit** Set to configure the module x as an 8-bit Pulse Width Modulator with output waveform on CEXx pin.
0	ECCFn	**Enable CCFx Interrupt bit** Clear to disable CCFx bit in CCON register to generate an interrupt request. Set to enable CCFx bit in CCON register to generate an interrupt request.

Reset Value = X000 0000b

Figure 5

PCA Capture Mode registers.

Example
Write a program in C to output a ramp function using PWM on bit P1.3. Run the program in the simulator and, using the logic analyzer, show the ramp function in PWM on P1.3.

Solution:
In the C-Code we need to set up three registers: CMOD, CCON, and CCAPM0.

CMOD is shown in Figure 2. Bits 2 and 1 (CPS1 and CPS0) control the PCA timer source. If these two bits are set to 01 the timer source will be the CPU clock/2. We want CMOD = 0x02;

CCON is the PCA control register shown in Figure 4. In this register we need to turn on the PCA timer which is bit 6. We want CCON = 0x40;

CCAPM0 is the capture mode register shown in Figure 5. In this register we want to enable the compare mode (bit 6) and enable PWM (bit 1). We want CCAPM0 = 0x42.

The remainder of the program is an infinite loop which calculates the data value for a ramp function and sends it periodically to the high byte of the capture register CCAPH0. If we use the variable pcaData as the signal variable we want CCAPH0 = pcaData;

To calculate a ramp signal we need only declare pcaData to be an unsigned char and increment it forever. We will also need a delay function to determine how often we will update the ramp function. For this example we will use a simple software delay loop that runs about 250 μsec.

```
//CPWM.c
// This program produces a ramp function on the PWM port bit P1.3.
#include<AT89C51CC03.h>
void Delay(int x);
void main(void)
   {unsigned char pcaData; //data byte for the ramp function
    CKCON = 0x01;   // x2 mode
    CMOD = 0x02;    //Clock bit CPS1, CPS0 = 01 so PCA clock
                    //  is cpu clock/2
    CCON = 0x40;    //Bit 6 in CCON turns on PCA timer
    CCAPM0 = 0x42;  //Bit 6 enables the compare mode for PWM
                    //  and bit 2 enables PWM
    while(1)
      {CCAP0H = pcaData;  //Load data to high byte of capture register
       pcaData++;         //Increment register for a ramp
       Delay(1);         //Software delay for about 250 useconds
      }
  }
void Delay(int x)
  {int i, j;
   for(i=0;i<x;i++)
     for(j=0;j<10;j++);
   return;
  }
```

Figure 6
A C program to output a ramp function on P1.3 as a PWM signal.

To view the output on the logic analyzer in the Keil IDE copy the code to a project and compile it. Set the crystal frequency to 28.2076 MHz. Open the debugger and use the view menu to open the logic analyzer. Click on Setup. In the setup window push the Insert key on the keyboard to insert a new signal. For the signal name enter `port1.3` for the signal name and push Enter. The set up window will look like that shown in Figure 7. Note the Keil enter Port1 for the signal name an puts in an AND mask to mask off bit 3.

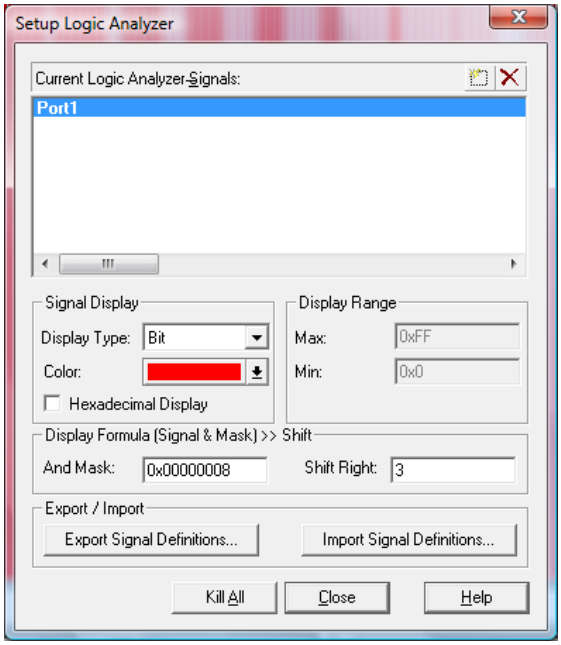

Figure 7
Setup window for creating a signal on P1.3.

Close the setup window and run the simulation for a few seconds. Turn the simulation off and your results will look like that shown in figure 8.

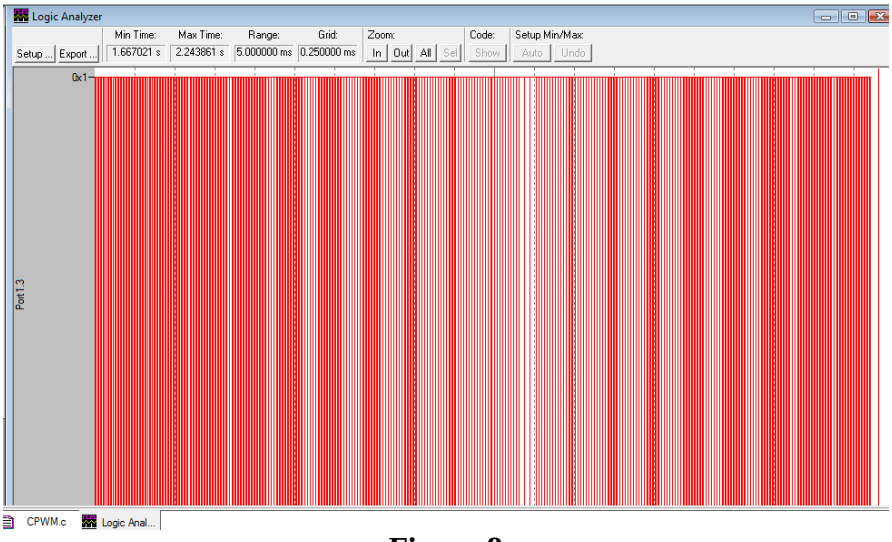

Figure 8
A simulated PWM ramp function on P1.3.