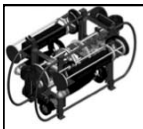
IMDL Software Series

Atmel AVR Xmega Code using GPIO from the Atmel ASF for PWM on DAQ Boards

A. A. Arroyo



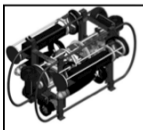
IMDL Software Series

16-Bit Timer/Counter

- Atmel AVR XMEGA devices have a set of flexible, 16-bit timer/counters (TC). Their capabilities include accurate program execution timing, frequency and waveform generation, and input capture with time and frequency measurement of digital signals.

Features

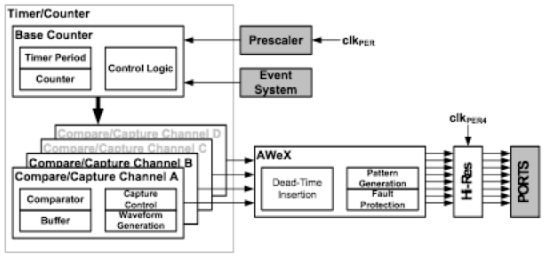
- 16-bit timer/counter
- 32-bit timer/counter support by cascading two timer/counters
- Up to four compare or capture (CC) channels
  - Four CC channels for timer/counters of type 0
  - Two CC channels for timer/counters of type 1
- Double buffered timer period setting
- Double buffered capture or compare channels
- Waveform generation:
  - Frequency generation
  - Single-slope pulse width modulation
  - Dual-slope pulse width modulation
- Input capture:
  - Input capture with noise cancelling
  - Frequency capture
  - Pulse width capture
  - 32-bit input capture
- Timer overflow and error interrupts/events
- One compare match or input capture interrupt/event per CC channel
- Can be used with event system for:
  - Quadrature decoding
  - Count and direction control
  - Capture
- Can be used with DMA and to trigger DMA transactions
- High-resolution extension
  - Increases frequency and waveform resolution by 4x (2-bit) or 8x (3-bit)
- Advanced waveform extension:
  - Low- and high-side output with programmable dead-time insertion (DTI)
  - Event controlled fault protection for safe disabling of drivers



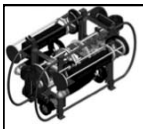
IMDL Software Series

16-Bit Timer/Counter

Figure 14-1. 16-bit timer/counter and closely related peripherals.



- A timer/counter consists of a base counter and a set of compare or capture (CC) channels. The base counter can be used to count clock cycles or events. It has direction control and period setting that can be used for timing. The CC channels can be used together with the base counter to do compare match control, frequency generation, and pulse width waveform modulation, as well as various input capture fcns.

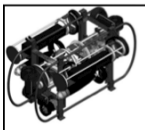


IMDL Software Series

16-Bit Timer/Counter

•Compare and capture cannot be done at the same time, i.e. a single Timer/Counter cannot simultaneously perform both waveform generation and capture operation. When used for compare operations, the CC channels is referred to as compare channels. When used for capture operations, the CC channels are referred to as capture channels.

•The Timer/Counter comes in two versions: Timer/Counter 0 that has four CC channels, and Timer/Counter 1 that has two CC channels. Hence, all registers and register bits that are related to CC channel 3 and CC channel 4 will only exist in Timer/Counter 0.

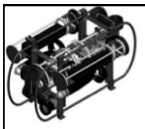


IMDL Software Series
16-Bit Timer/Counter

Table 14-1. Timer/counter definitions.

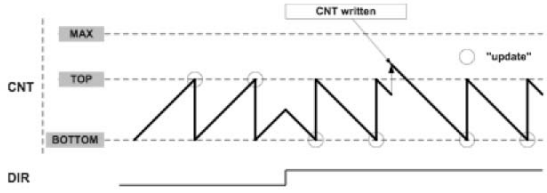
Name	Description
BOTTOM	The counter reaches BOTTOM when it becomes zero.
MAX	The counter reaches MAXimum when it becomes all ones.
TOP	The counter reaches TOP when it becomes equal to the highest value in the count sequence. The TOP value can be equal to the period (PER) or the compare channel A (CCA) register setting. This is selected by the waveform generator mode.
UPDATE	The timer/counter signals an update when it reaches BOTTOM or TOP, depending on the waveform generator mode.

- The Counter Register (CNT), the Period Registers w/buffer (PER and PERBUF), and the compare and Capture registers w/buffers (CCx and CCxBUF) are 16-bit registers.
- During normal operation the counter value is continuously compared to zero and the period (PER) value to determine whether the counter has reached TOP or BOTTOM. The counter value is also compared to the CCx registers. These comparators can be used to generate interrupt requests or request DMA transactions. They also generate events for the Event System.

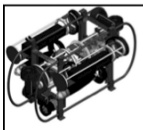


IMDL Software Series
16-Bit Timer/Counter

Figure 14-6. Normal operation.



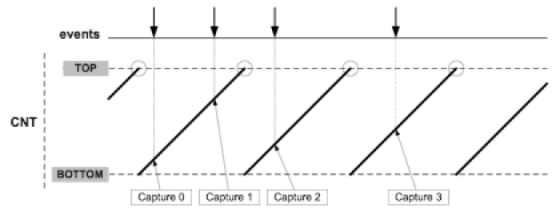
- In Normal Operation the Counter will count in the direction set by the Direction (DIR) bit for each clock until it reaches TOP or BOTTOM. When TOP is reached when up-counting, the counter will be set to zero when the next clock is given. When down-counting the Counter is reloaded with Period Register value when BOTTOM is reached.



IMDL Software Series

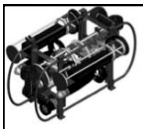
16-Bit Timer/Counter

Figure 14-11.Input capture timing.



•Selecting the input capture event action, makes the enabled capture channel perform an input capture on any event. The interrupt flags will be set and indicate that there is a valid capture result in the corresponding CC register. Equally the buffer valid flags indicates valid data in the buffer registers.

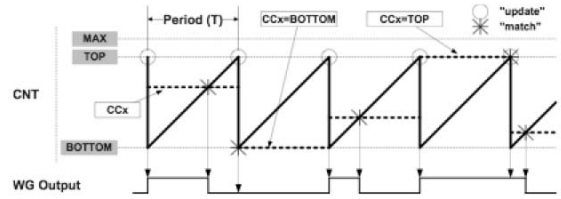
•The counter will continuously count for BOTTOM to TOP, then restart on BOTTOM as shown in Figure 14-9. The figure also shows four capture events for one capture channel.



IMDL Software Series

Single-Slope PWM Mode

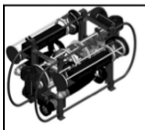
Figure 14-15.Single-slope pulse width modulation.



•Single-Slope Mode: For ssPWM generation, the Period (T) is controlled by the PER, while CCx registers control the duty cycle of the WG output. Figure 14-13 shows how the counter counts from BOTTOM to TOP then restarts from BOTTOM. The waveform generator output is set on the compare match between the CNT and CCx registers, and cleared at TOP.

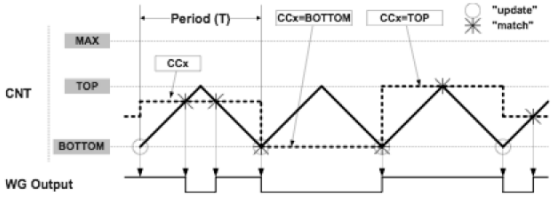
•The single slow PWM frequency (f_{PWM_SS}) depends on the period setting (PER) and the Peripheral clock frequency (f_{PER}), and can be calculated by the following formula where N is the prescaler divider used (1, 2, 4, 8, 64, 256, 1024).

$$f_{PWM\_SS} = \frac{f_{PER}}{N(PER + 1)}$$



IMDL Software Series
Dual-Slope PWM Mode

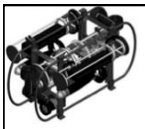
Figure 14-16.Dual-slope pulse width modulation.



•For dual slope PWM generation, the Period (T) is controlled by the PER, while CCx registers control the duty cycle of the WG output. Figure 14-14 shows how for dual slope PWM the Counter counts repeatedly from BOTTOM to TOP, and then from TOP to BOTTOM. The waveform generator output is set on BOTTOM, cleared on compare match when up-counting and set on compare match when down counting.

•The PWM frequency depends on the period setting (PER) and the Peripheral Clock frequency (f_{PER}), and can be obtained by the equation (f_{PWM_DS}) where N is the prescaler divider used (1, 2, 4, 8, 64, 256, 1024).

$$f_{PWM\_DS} = \frac{f_{PER}}{2 \cdot N \cdot PER}$$



IMDL Software Series
PWM Details

- The PWM clock is selected in the CTRLA register. Example:

  TCC0_CTRLA = 0x05; //set TCC0_CLK to CLK/64

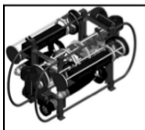
CTRLA – Control register A

Bit	7	6	5	4	3	2	1	0
+0x00	–	–	–	–	CLKSEL[3:0]			
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 3:0 – CLKSEL[3:0]: Clock Select**
  These bits select the clock source for the timer/counter according to Table 14-3.
  CLKSEL=0001 must be set to ensure a correct output from the waveform generator when the hi-res extension is enabled.

Table 14-3. Clock select options.

CLKSEL[3:0]	Group configuration	Description
0000	OFF	None (i.e, timer/counter in OFF state)
0001	DIV1	Prescaler: Clk
0010	DIV2	Prescaler: Clk/2
0011	DIV4	Prescaler: Clk/4
0100	DIV8	Prescaler: Clk/8
0101	DIV64	Prescaler: Clk/64
0110	DIV256	Prescaler: Clk/256
0111	DIV1024	Prescaler: Clk/1024
1nnn	EVCHn	Event channel n, n= [0,...,7]



IMDL Software Series
PWM Details

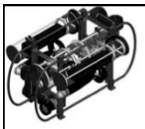
- PWM WG is selected in the CTRLB reg. Example:
  // Enable OC A, B, C, and D. Set to Single Slope PWM
  // OCnX = 1 from Bottom to CCx and 0 from CCx to Top.
  TCC0_CTRLB = 0xF3;

Bit	7	6	5	4	3	2	1	0
+0x01	CCDEN	CCCEN	CCBEN	CCAEN	–	WGMODE[2:0]		
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:4 – CCxEN: Compare or Capture Enable**
  Setting these bits in the FRQ or PWM waveform generation mode of operation will override the port output register for the corresponding OCn output pin.
  When input capture operation is selected, the CCxEN bits enable the capture operation for the corresponding CC channel.
- **Bit 2:0 – WGMODE[2:0]: Waveform Generation Mode**
  These bits select the waveform generation mode, and control the counting sequence of the counter, TOP value, UPDATE condition, interrupt/event condition, and type of waveform that is generated according to Table 14-4 on page 175.
  No waveform generation is performed in the normal mode of operation. For all other modes, the result from the waveform generator will only be directed to the port pins if the corresponding CCxEN bit has been set to enable this. The port pin direction must be set as output

Table 14-4. Timer waveform generation mode.

WGMODE[2:0]	Group configuration	Mode of operation	Top	Update	OVFIF/Event
000	NORMAL	Normal	PER	TOP	TOP
001	FRQ	Frequency	CCA	TOP	TOP
010		Reserved	–	–	–
011	SINGLESLOPE	Single-slope PWM	PER	BOTTOM	BOTTOM
100		Reserved	–	–	–
101	DSTOP	Dual-slope PWM	PER	BOTTOM	TOP
110	DSBOTH	Dual-slope PWM	PER	BOTTOM	TOP and BOTTOM
111	DSBOTTOM	Dual-slope PWM	PER	BOTTOM	BOTTOM



IMDL Software Series
PWM Details

- The PER holds the value TOP. Example: PER = Top
  TCC0_PER = 10000; // 20ms x (32MHz/64) = 10000
  {If we had the option to divide by 32 instead of 64 then TCC0_PER would have been 20000 for 20ms duration instead of 10000. To get a 0.5ms pulse we would have used TCC0_CCA=500 instead of having to use TCC0_CCA=250 which is 50% of the value… not too bad}

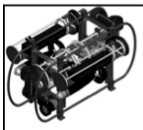
The PERH and PERL register pair represents the 16-bit value, PER. PER contains the 16-bit TOP value in the timer/counter.

Bit	7	6	5	4	3	2	1	0
+0x26	PER[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	1

- **Bit 7:0 – PER[7:0]: Periodic low byte**
  These bits hold the LSB of the 16-bit period register.

Bit	7	6	5	4	3	2	1	0
+0x27	PER[15:8]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	1

- **Bit 7:0 – PER[15:8]: Periodic high byte**
  These bits hold the MSB of the 16-bit period register.



IMDL Software Series
PWM Details

- The CCxH and CCxL register pair represents the 16-bit value CCx. For compare operation these registers are all continuously compared to the counter value. Normally the outputs form the comparators are then used for generating waveforms. Example:

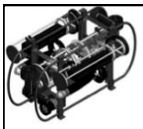
```
TCC0_CCA = 0;          //PWMC0 off.
TCC1_CCA = 0;          //PWMC4 off
```

Bit	7	6	5	4	3	2	1	0
	CCx[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:0 – CCx[7:0]: Compare or Capture x low byte**
  These bits hold the LSB of the 16-bit compare or capture register.

Bit	7	6	5	4	3	2	1	0
	CCx[15:8]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

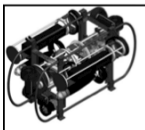
- **Bit 7:0 – CCx[15:8]: Compare or Capture x high byte**
  These bits hold the MSB of the 16-bit compare or capture register.



IMDL Software Series
PWM Details

- To generate ssPWM we can set CCx to the desired period directly. To make it more useful, we can scale a -100 < input period < +100 as follows. Example:

```
void ServoC(int channel, int value) {
  if (value > 100)          // cap at +/- 100
     value = 100;           //   -100 => 1ms
  else if (value < -100)    //   0    => 1.5ms
     value = -100;          //   100  => 2ms
  value *= 5;               // multiply value by 2.5
  value /= 2;               // new range +/- 250
  if (channel==0) TCC0_CCA=(750+value); //Generate C0 PWM
}
```



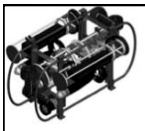
IMDL Software Series

* AAA_Util.c - Notepad2 (Administrator)

File Edit View Settings ?

```
28 void PWMCInit(void) {                 // PWMCINIT: Initialize PWM Channels on PortC
29     TCC0_CTRLA   = 0x05;             // set TCC0_CLK to CLK/64
30     TCC0_CTRLB   = 0x33;             // Enable OC A, B.  Set to Single Slope PWM
31                                      // OCnX = 1 from Bottom to CCx and 0 from CCx to
32     TCC0_PER     = 10000;            // 20ms / (1/(32MHz/64)) = 10000.  PER = Top
33     TCC1_CTRLA   = 0x05;             // set TCC1_CLK to CLK/64
34     TCC1_CTRLB   = 0x33;             // Enable OC A and B.  Set to Single Slope PWM
35                                      // OCnX = 1 from Bottom to CCx and 0 from CCx to
36     TCC1_PER     = 10000;            // 20ms / (1/(32MHz/64)) = 10000.  PER = Top
37     PORTC_DIRSET = 0x33;             // set PORTC5:4,1:0 to output
38 //  PORTC_DIR    = 0x33;             // set PORTC5:4,1:0 to output
39     TCC0_CCA     = 0;                // PWMC0 off
40     TCC0_CCB     = 0;                // PWMC1 off
41 //  TCC0_CCC     = 0;                // PWMC2 off
42 //  TCC0_CCD     = 0;                // PWMC3 off
43     TCC1_CCA     = 0;                // PWMC4 off
44     TCC1_CCB     = 0;                // PWMC5 off
45 }   // End PWMCINIT
46
47 void PWMC(int channel, int value) {            // PWMC: Generate PWM on PortC pin
48     int ledval, serval;
49     if (value > 100) value = 100;               // cap at +/- 100
50     else if (value < -100) value = -100;        // -100 => 1ms  0 => 1.5ms  100 =>
51     ledval=value;
52     serval=value;
53     serval *= 5;                                // multiply value by 5 {avoid floa
54     ledval *= 50;                               // new range 0<range<10000  for LE
55     if (channel==0) TCC0_CCA = (5000 + ledval); // Generate C0 PWM for LEDs.
56     if (channel==1) TCC0_CCB = (5000 + ledval); // Generate C1 PWM for LEDs.
57 //  if (channel==0) TCC0_CCA = (750 + serval);  // Generate C0 PWM for Servos.
58 //  if (channel==1) TCC0_CCB = (750 + serval);  // Generate C1 PWM for Servos.
59 //  if (channel==2) TCC0_CCC = (750 + serval);  // Generate C2 PWM for Servos.
60 //  if (channel==3) TCC0_CCD = (750 + serval);  // Generate C3 PWM for Servos.
61     if (channel==4) TCC1_CCA = (600 + serval);  // Generate C4 PWM for Servos.
62     if (channel==5) TCC1_CCB = (750 + serval);  // Generate C5 PWM for Servos.
63 }   // end PWMC
```

Ln 31 : 85 Col 1 Sel 0 | 4.00 KB | ANSI | CR+LF INS C/C++ Source Code



IMDL Software Series

AAA_Util.c - Notepad2 (Administrator)

File Edit View Settings ?

```
28 void PWMDInit(void) {                 // PWMDINIT: Initialize PWM Channels on PortD
29     TCD0_CTRLA   = 0x05;             // set TCD0_CLK to CLK/64
30     TCD0_CTRLB   = 0xF3;             // Enable OC A, B, C, and D.  Set to Single Sl
31                                      // OCnX = 1 from Bottom to CCx and 0 from CCx t
32     TCD0_PER     = 10000;            // 20ms / (1/(32MHz/64)) = 10000.  PER = Top
33     TCD1_CTRLA   = 0x05;             // set TCD1_CLK to CLK/64
34     TCD1_CTRLB   = 0x33;             // Enable OC A and B.  Set to Single Slope PWM
35                                      // OCnX = 1 from Bottom to CCx and 0 from CCx t
36     TCD1_PER     = 10000;            // 20ms / (1/(32MHz/64)) = 10000.  PER = Top
37     PORTD_DIRSET = 0x3F;             // set PORTD pins <5:0> to output
38 //  PORTCD_DIR   = 0x3F;             // set PORTD pins <5:0> to output
39     TCD0_CCA     = 0;                // PWMD0 off
40     TCD0_CCB     = 0;                // PWMD1 off
41     TCD0_CCC     = 0;                // PWMD2 off
42     TCD0_CCD     = 0;                // PWMD3 off
43     TCD1_CCA     = 0;                // PWMD4 off
44     TCD1_CCB     = 0;                // PWMD5 off
45 }   // End PWMDINIT
46
47 void PWMD(int channel, int value) {            // PWMD: Generate PWM on PortC
48     int ledval, serval;
49     if (value > 100) value = 100;               // cap at +/- 100
50     else if (value < -100) value = -100;        // -100% => 0.5ms  0% => 1.5ms
51     ledval=value;
52     serval=value;
53     serval *= 5;                                // multiply value by 5 {avoid f
54     ledval *= 50;                               // new range 0<range<10000  for
55     if (channel==0) TCD0_CCA = (5000 + ledval); // Generate D0 PWM for LEDs.
56     if (channel==1) TCD0_CCB = (5000 + ledval); // Generate D1 PWM for LEDs.
57 //  if (channel==0) TCD0_CCA = (750 + serval);  // Generate D0 PWM for Servos.
58 //  if (channel==1) TCD0_CCB = (750 + serval);  // Generate D1 PWM for Servos.
59     if (channel==2) TCD0_CCC = (750 + serval);  // Generate D2 PWM for Servos.
60     if (channel==3) TCD0_CCD = (750 + serval);  // Generate D3 PWM for Servos.
61     if (channel==4) TCD1_CCA = (750 + serval);  // Generate D4 PWM for Servos.
62     if (channel==5) TCD1_CCB = (750 + serval);  // Generate D5 PWM for Servos.
63 }   // end PWMD
```

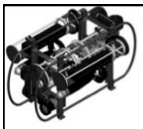
IMDL Software Series
PWM

- The next example uses the ADC in the DAQ to sample a Cds cell and turn on/off the dbled in the main loop and uses PWM in PD0 and PD1 to gradually dim the external LEDs connected to PD0 and PD1 & generates PWM servo signals in PD2-PD5

See project DAQ_GPIO_PWM

- If you wish to use Tim's USART functions, then import uart.c and uart.h from Tim's Software
- The ISR for the USART is uart.c

See project DAQ_GPIO_PWM
Change the include in main.c to <uart.h>



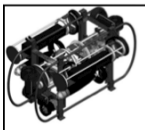
IMDL Software Series

main.c - Notepad2 (Administrator)

File Edit View Settings ?

```
#include <asf.h>
#include <avr/io.h>
#include <stdio.h>
#include <avr/interrupt.h>
//#include "usart.h"
#include "uart.h"
#include "adc.h"
#include "AAA_Util.h"
#include "clock.h"


int main(void) {

    clockInit();

    port_pin_t      PC0 = IOPORT_CREATE_PIN(PORTC,0);
    port_pin_t      PD0 = IOPORT_CREATE_PIN(PORTD,0);
    port_pin_t      PD1 = IOPORT_CREATE_PIN(PORTD,1);
//  port_id_t       PD  = IOPORT_CREATE_ID(PORTD);

    PORTC_DIRSET =  0x01;           //Debug Led
    PORTD_DIRSET =  0x03;           //External LEDs on PD0 & PD1
//  usart_Init();
    uartInit(&USARTD1,57600); //Initialize uart connected to the USB port.
    stdout = &USB_str; //stdout --> USB : printf writes to the USB port.
    sei();

    printf("DAQ LEDs, PWM, ADC & USART test\r");
    printf("Enter i to turn on DLED or o to turn it off\r");
    printf("Enter j to turn on PD0 or k to turn on PD1\r");
    printf("? \n");

    int16_t  m=0;

    for (m=0;m<3;m++) { // number of times to blink LEDs
        printf("LED Blink m= %d\r",m);
```

Ln 1 : 134 Col 1 Sel 0 4.33 KB ANSI CR+LF INS C/C++ Source Code



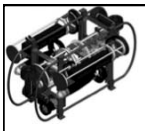
IMDL Software Series

main.c - Notepad2 (Administrator)

File Edit View Settings ?

```
65          gpio_set_pin_low(PC0);
66  //      ioport_set_pin_low(PC1);
67  //      PORTC_OUTCLR = 0x02;    // Turns the debug led on.  The led is active low
68          gpio_set_pin_high(PD0);
69          gpio_set_pin_low(PD1);
70  //      ioport_set_pin_high(PD0);
71  //      ioport_set_pin_low(PCD1);
72  //      ioport_set_group_high(PD,0x01);
73  //      ioport_set_group_low(PD,0x02);
74  //      PORTD_OUT = 0x01;       // turn on External LED on PD0, off LED on PD1
75          delay_ms(500);          // built-in delay function
76
77          gpio_set_pin_high(PC0);
78  //      ioport_set_pin_high(PC1);
79  //      PORTC_OUTSET = 0x02;    // Turns the debug led off.  The led is active low
80          gpio_set_pin_low(PD0);
81          gpio_set_pin_high(PD1);
82  //      ioport_set_pin_low(PD0);
83  //      ioport_set_pin_high(PD1);
84  //      ioport_set_group_high(PD,0x02);
85  //      ioport_set_group_low(PD,0x01);
86  //      PORTD_OUT = 0x02;       // turn off External LED on PD0, on LED on PD1
87          delay_ms(500);          // built-in delay function }
88
89      } // end for loop
90
91      gpio_set_pin_low(PC0);
92  //  ioport_set_pin_low(PC1);
93  //  PORTC_OUTCLR = 0x02;    // Turns the debug led on.  The led is active low
94      gpio_set_pin_high(PD0);
95      gpio_set_pin_high(PD1);
96  //  ioport_set_group_high(PD,0x03);
97  //  PORTD_OUT = 0x03;       // turn on External LEDs on PD0, and LED on PD1
98      PWMDInit();                     // setup PORTD PWM Channels
99      adcInit();                      // setup PORTA analog channels
100     delay_ms(100);                  // wait for ADC to be ready
```

Ln 1 : 134 Col 1 Sel 0 | 4.33 KB | ANSI | CR+LF INS | C/C++ Source Code



IMDL Software Series

* main.c - Notepad2 (Administrator)

File Edit View Settings ?

```
101     int16_t A1,ambient = analogRead(0); // Read Cds connected to ADC Ch0
102     printf("Ambient:%d\r",ambient);
103
104     while(1) {              // infinite Loop
105         for(m=-100;m<101;m+=5) {
106             if ( m % 20 == 0 ){
107                 A1 = analogRead(0);
108                 printf("Ambient:%d   A1=%d   Servo Percent=%d\r",ambient,A1,m);
109                 if (A1 > 1.1*ambient ) gpio_set_pin_high(PC0);  // if more light turn off D
110                 else gpio_set_pin_low(PC0);                     // else turn on DBLED
111                 }  // End check the Cds cell
112             PWMD(0,m);
113             PWMD(1,-m);
114             PWMD(2,m);
115             PWMD(3,-m);
116             delay_ms(100);
117
118
119         }   // End sweep PWM {for loop}
120         PWMD(2,0);
121         PWMD(3,0);
122         delay_ms(750);
123     }   // End of infinite while loop
124
125 } // end main
126
127 ISR(USARTD1_RXC_vect){
128     uint8_t rcvData = 0;                                //AAA
129     rcvData = USARTD1.DATA;                             //AAA
130     if (rcvData == 'i') PORTC_OUTCLR = 0x01;            //AAA
131     else if (rcvData == 'o') PORTC_OUTSET = 0x01;       //AAA
132     else if (rcvData == 'j') {PORTD_OUTSET = 0x01; PORTD_OUTCLR=0x02;}     //AAA
133     else if (rcvData == 'k') {PORTD_OUTSET = 0x02; PORTD_OUTCLR=0x01;}     //AAA
134     storeD1(rcvData);                                   //AAA storeD1(USARTD1.DATA)
135 }
136
```

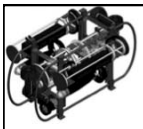
Ln 136 : 136 Col 1 Sel 0 | 4.34 KB | ANSI | CR+LF INS | C/C++ Source Code

IMDL Software Series

All in C

- The next example is identical to the previous example. It uses the ADC in the DAQ to sample a Cds cell and turn on/off the dbled in the main loop and uses PWM in PD0 and PD1 to gradually dim the external LEDs connected to PD0 and PD1 & generates PWM servo signals in PD2-PD5. It does not use the ASF (Atmel Software Framework). It is a GNU C project built from scratch.

See project DAQ_All_in_C



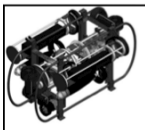
IMDL Software Series

* main.c - Notepad2 (Administrator)

File Edit View Settings ?

```
29 #include <asf.h>
30 #include <avr/io.h>
31 #include <stdio.h>
32 #include <avr/interrupt.h>
33 #include <util/delay.h>
34 //#include "usart.h"
35 #include "uart.h"
36 #include "adc.h"
37 #include "AAA_Util.h"
38 #include "clock.h"
39
40 int main(void) {
41     clockInit();
42     PORTC_DIRSET =  0x01;           //Debug Led
43     PORTD_DIRSET =  0x03;           //External LEDs on PD0 & PD1
44 //  usart_Init();
45     uartInit(&USARTD1,57600); //Initialize uart connected to the USB port.
46     stdout = &USB_str; //stdout --> USB : printf writes to the USB port.
47     sei();
48     printf("DAQ All in C\r");
49     printf("Enter i to turn on DLED or o to turn it off\r");
50     printf("Enter j to turn on PD0 or k to turn on PD1\r");
51     printf("? \n");
52
53     int16_t  m=0;
54     for (m=0;m<3;m++) { // number of times to blink LEDs
55         printf("LED Blink m= %d\r",m);
56         PORTC_OUTCLR = 0x02;     // Turns the debug led on.  The led is active l
57         PORTD_OUT = 0x01;        // turn on External LED on PD0, off LED on PD1
58         _delay_ms(500);          // built-in delay function
59         PORTC_OUTSET = 0x02;     // Turns the debug led off.  The led is active
60         PORTD_OUT = 0x02;        // turn off External LED on PD0, on LED on PD1
61         _delay_ms(500);          // built-in delay function }
62     } // end for loop
63
64     PORTC_OUTCLR = 0x02;     // Turns the debug led on.  The led is active low
```

Ln 48 : 100 Col 5 Sel 0 3.50 KB ANSI CR+LF INS C/C++ Source Code



IMDL Software Series

* main.c - Notepad2 (Administrator)

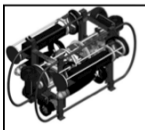
File Edit View Settings ?

```
65    PORTD_OUT = 0x03;        // turn on External LEDs on PD0, and LED on PD1
66    PWMDInit();              // setup PORTD PWM Channels
67    adcInit();               // setup PORTA analog channels
68    _delay_ms(100);          // wait for ADC to be ready
69    int16_t A1,ambient = analogRead(0); // Read Cds connected to ADC Ch0
70    printf("Ambient:%d\r",ambient);
71
72    while(1) {             // infinite Loop
73        for(m=-100;m<101;m+=5) {
74            if ( m % 20 == 0 ){
75                A1 = analogRead(0);
76                printf("Ambient:%d   A1=%d   Servo Percent=%d\r",ambient,A1,m);
77                if (A1 > 1.1*ambient ) PORTC_OUTSET = 0x02; // if more light turn off DBLED
78                else PORTC_OUTCLR = 0x02;                   // else turn on DBLED
79                } // End check the Cds cell
80            PWMD(0,m);
81            PWMD(1,-m);
82            PWMD(2,m);
83            PWMD(3,-m);
84            _delay_ms(100);
85        }  // End sweep PWM {for loop}
86        PWMD(2,0);
87        PWMD(3,0);
88        _delay_ms(750);
89    }  // End of infinite while loop
90 } // end main
91
92 ISR(USARTD1_RXC_vect){
93    uint8_t rcvData = 0;                          //AAA
94    rcvData = USARTD1.DATA;                       //AAA
95    if (rcvData == 'i') PORTC_OUTCLR = 0x01;      //AAA
96    else if (rcvData == 'o') PORTC_OUTSET = 0x01; //AAA
97    else if (rcvData == 'j') {PORTD_OUTSET = 0x01; PORTD_OUTCLR=0x02;}    //AAA
98    else if (rcvData == 'k') {PORTD_OUTSET = 0x02; PORTD_OUTCLR=0x01;}    //AAA
99    storeD1(rcvData);                             //AAA storeD1(USARTD1.DATA)
100 }
```

Ln 84 : 100 Col 28 Sel 0 | 3.50 KB | ANSI | CR+LF INS | C/C++ Source Code

IMDL Software Series
Arduino Servo

- The example uses the ADC functions in the Arduino MEGA2560 Board to blink internal/ external LEDs and uses the serial port to display data and control the LEDs after blinking the set a few times. Serial input *j* & *k* alternate the two external LEDs on pins 12 & 11 {on, off} and {off, on} respectively. Analog input A0 is connected to a Cds cell that controls the internal DBLED. A servo connected to pin 9 is also swept.
  - See Arduino Sketch: IMDL_Servo_ADC_Serial_External_LED



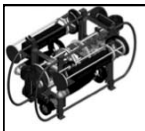
IMDL Software Series

IMDL_Servo_ADC_Serial_External_LED.c - Notepad2 (Administrator)

File Edit View Settings ?

```
/*
  EEL-5666 Example 4: Use of Digital Output Pins
  Turn on/off External LEDs connected to Pins 11 & 12
  alternatively and the internal LED on Pin 13 on/off.
  Add a delay in each state of 1/2 sec. Use serial input
  to control the LEDs. {i,o} for internal LED {j,k} for
  external LEDs. Read Cds cell on Analog A0 and control
  the internal LED. Use pin 9 to sweep a servo.
  Arroyo September 10 2014

*/

#include <Servo.h>
Servo myservo;  // create servo object to control a servo

int extLED1 = 11;   // External LED connected to digital pin 11
int extLED2 = 12;   // External LED connected to digital pin 12
int intLED  = 13;   // Internal LED connected to digital pin 13

void setup( ) {
  pinMode(extLED1, OUTPUT);     // sets digital pin 12 as output
  pinMode(extLED2, OUTPUT);     // sets digital pin 12 as output
  pinMode(intLED,  OUTPUT);     // sets digital pin 13 as output
  Serial.begin(57600);          // connect to the serial port
  myservo.attach(9);            // attaches servo to pin 9
}

  int rcvData=0, count=0, j=0, a1, ambient;
void loop( ) {
  if (count < 1) {
    Serial.println("Arduino MEGA2560 Example");
    Serial.println("Cds Cell connected on Analog A0");
    Serial.println("Control the internal LED on pin 13");
    ambient = analogRead(A0);
    Serial.print("Ambient: ");
    Serial.println(ambient);
```

Ln 1:87 Col 1 Sel 0 | 3.33 KB | ANSI | LF | INS | C/C++ Source Code



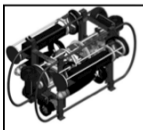
IMDL Software Series

IMDL_Servo_ADC_Serial_External_LED.c - Notepad2 (Administrator)

File Edit View Settings ?

```
    for (j=0; j<5; j++){
      Serial.print("j= ");
      Serial.println(j);
      digitalWrite(13, HIGH);   // set the internal LED on
      digitalWrite(12, LOW);    // set the external LED1 off
      digitalWrite(11, HIGH);   // set the external LED2 on
      delay(500);               // wait for a 1/2 second
      digitalWrite(13, LOW);    // set the internal LED off
      digitalWrite(12, HIGH);   // set the external LED on
      digitalWrite(11, LOW);    // set the external LED off
      delay(500);               // wait for a 1/2 second
    }
    digitalWrite(13, HIGH);   // set the internal LED  on
    digitalWrite(12, HIGH);   // set the external LED1 on
    digitalWrite(11, HIGH);   // set the external LED2 on
  }
  count=1;
  Serial.print("Ambient: ");
  Serial.print(ambient);
  Serial.print("   A1: ");
  a1 = analogRead(A0);
  if (a1 > 1.1*ambient ) digitalWrite(13, LOW); // turn off DBLED
  else digitalWrite(13, HIGH);    // else turn on DBLED
  Serial.println(a1);

//  Sweep Servo
  for(j = 0; j < 180; j += 1) {    // goes from 0 degrees to 180 degrees
    myservo.write(j);              // tell servo to go to position in variable 'pos'
    delay(15);                     // waits 15ms for the servo to reach the position
  }
  for(j = 180; j >= 1; j -= 1) {   // goes from 180 degrees to 0 degrees
    myservo.write(j);              // tell servo to go to position in variable 'pos'
    delay(15);                     // waits 15ms for the servo to reach the position
  }
```

Ln 1:87 Col 1 Sel 0 | 3.33 KB | ANSI | LF | INS | C/C++ Source Code



IMDL Software Series

```
72
73  // blink data only when you receive data:
74  if (Serial.available() > 0) {
75  // read the incoming byte:
76     rcvData = Serial.read();
77  // say what you got:
78     Serial.print("I received: ");
79     Serial.write(rcvData);
80     Serial.println();
81     if (rcvData == 'j') {digitalWrite(12,LOW); digitalWrite(11,HIGH);}
82     else if (rcvData == 'k') {digitalWrite(12,HIGH); digitalWrite(11,LOW);}
83 //     else if (rcvData == 'i') digitalWrite(13, HIGH);      // internal LED on
84 //     else if (rcvData == 'o') digitalWrite(13, LOW);  // internal LED off
85     }
86 }
87
```

Ln 1 : 87 Col 1 Sel 0 | 3.33 KB | ANSI | LF | INS | C/C++ Source Code

IMDL Software Series

The End!