DXCORE - TCA0Demo, TCA0Demo2, TCA0Demo3, TCA0Demo4

Example Code

```
/* Example 1: 16-bit PWM in single mode, dual slope with interrupt.
 *
https://github.com/SpenceKonde/DxCore/blob/master/megaavr/extras/TakingOverTCA0.md
 *
 * The whole "ISR adjusting duty cycle" like that probably isn't something you'd
actually
 * want to do - but it demonstrates how to configure an ISR on TCA0, which is the
point.
 */

#if defined(MILLIS_USE_TIMERA0)
  #error "This sketch takes over TCA0, don't use for millis here."
#endif

unsigned int DutyCycle = 0;
// picked more or less randomly, other than the fact that everything has it, so it
makes a good example :-)
uint8_t OutputPin = PIN_PC1;


void setup() {
  pinMode(OutputPin, OUTPUT);
  takeOverTCA0(); // this replaces disabling and resettng the timer, required
previously.
  PORTMUX.TCAROUTEA   = (PORTMUX.TCAROUTEA & ~(PORTMUX_TCA0_gm)) |
PORTMUX_TCA0_PORTC_gc; // Set mux to PORTC
  TCA0.SINGLE.CTRLB   = (TCA_SINGLE_CMP1EN_bm | TCA_SINGLE_WGMODE_DSBOTTOM_gc); //
Dual slope PWM mode OVF interrupt at BOTTOM, PWM on WO1.
  TCA0.SINGLE.PER     = 0xFFFF;               // Count all the way up to 0xFFFF.
  //                                             At 20MHz, this gives ~152Hz PWM
with no prescaling.
  TCA0.SINGLE.CMP1    = DutyCycle;            // 0 - 65535
  TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;    // enable overflow interrupt
  TCA0.SINGLE.CTRLA   = TCA_SINGLE_ENABLE_bm; // enable the timer with no
prescaler
}

void loop() { // Not even going to do anything in here
}

ISR(TCA0_OVF_vect) {    // on overflow, we will increment TCA0.CMP0, this will
happen after every full cycle - a little over 7 minutes.
  TCA0.SINGLE.CMP1      = DutyCycle++; // Because we are in Dual Slope Bottom
mode, OVF fires at BOTTOM, at end, not TOP, in middle of the pulse.
```

```
  TCA0.SINGLE.INTFLAGS  = TCA_SINGLE_OVF_bm; // Always remember to clear the interrupt flags, otherwise the interrupt will fire continually!
}
```

```

/* Example 2: Variable frequency and duty cycle PWM
 * https://github.com/SpenceKonde/DxCore/blob/master/megaavr/extras/TakingOverTCA0.md
 *
 * This generates PWM similar to the first example (though without the silly interrupt to change the duty cycle),
 * but takes it a step further and into more practical territory with two functions to set the duty cycle and frequency.
 * Calling those instead of this PWMDemo() function is all you'd need to make use of this.
 * Somewhere I think I have the same functionality implemented for the classic AVR "Timer1" style 16-bit timers.
 */


#if defined(MILLIS_USE_TIMERA0)
  #error "This sketch takes over TCA0, don't use for millis here."
#endif

uint8_t OutputPin = PIN_PC0;

unsigned int Period = 0xFFFF;

void setup() {
  pinMode(OutputPin, OUTPUT);
  PORTMUX.TCAROUTEA = (PORTMUX.TCAROUTEA & ~(PORTMUX_TCA0_gm)) | PORTMUX_TCA0_PORTC_gc;
  takeOverTCA0(); // this replaces disabling and resettng the timer, required previously.
  TCA0.SINGLE.CTRLB = (TCA_SINGLE_CMP0EN_bm | TCA_SINGLE_WGMODE_SINGLESLOPE_gc); //Single slope PWM mode, PWM on WO0
  TCA0.SINGLE.PER   = Period; // Count all the way up to 0xFFFF; At 20MHz, no prescale, this gives ~305Hz PWM
  TCA0.SINGLE.CMP0  = 0;
  TCA0.SINGLE.CTRLA = TCA_SINGLE_ENABLE_bm; // Eable the timer with no prescaler
}

void loop() {
  PWMDemo(150000);  // 150kHz
  PWMDemo(70000);   // 70kHz
  PWMDemo(15000);   // 15kHz
  PWMDemo(3000);    // 3kHz
  PWMDemo(120);     // 120Hz
  PWMDemo(35);      // 35Hz
  PWMDemo(13);      // 13Hz
}

```

```
void PWMDemo(unsigned long frequency) {
  setFrequency(frequency);
  setDutyCycle(64);   // ~25%
  delay(4000);
  setDutyCycle(128);  // ~50%
  delay(4000);
  setDutyCycle(192);  // ~75%
  delay(4000);
}

void setDutyCycle(byte duty) {
  TCA0.SINGLE.CMP0 = map(duty, 0, 255, 0, Period);
  // map() kinda sucks, there are better ways to do this, etc. For more information, consult
  // a different guide written by somebody else. No, I don't have one in mind ;)
}

void setFrequency(unsigned long freqInHz) {
  unsigned long tempperiod = (F_CPU / freqInHz);
  byte presc = 0;
  while (tempperiod > 65536 && presc < 7) {
    presc++;
    tempperiod = tempperiod >> (presc > 4 ? 2 : 1);
  }
  Period = tempperiod;
  TCA0.SINGLE.CTRLA = (presc << 1) | TCA_SINGLE_ENABLE_bm;
  TCA0.SINGLE.PER = Period;
}
```

```
/* Example 3: High speed 8-bit PWM
 * https://github.com/SpenceKonde/DxCore/blob/master/megaavr/extras/TakingOverTCA0.md
 *
 * A user of megaTinyCore requested (#152) high speed PWM. They wanted split mode disabled, and PWM frequency higher
 * than 62KHz. This is indeed possible - though do note that the maximum frequency of PWM possible with a full 8 bits
 * of resolution is 78.125 kHz when running at 20 MHz (20000000/256); at 24, it's 93.75 kHz, and overclocked to 32 MHz,
 * 125 kHz. The next highest  frequency for which perfect 8-bit resolution is possible is half of those frequencies.
 * Higher fequencies require lower resolution (see above example for one approach, which can also be used for
 * intermediate frequencies) - though if the frequency is constant, varying your input between 0 and the period instead
 * of using map() is desirable, as map may not be smooth. As a further aside, if 78.125kHz is suitable, there is no
 * need to disable split mode (ynless other features were required, like event inputs or buffering (which might well
 * be what the original requester wanted single mode for)
 * It strikes me now, as I adapt this example for the Dx-series parts, that 62 KHz is almost exactly the maximum
 * possible for 8-bit PWM at 16 MHz system clock. I'm pretty sure there's a connection!
 *
 * Do note that if pushing the PWM frequency is your aim, you can go considerably higher by using the Type D timer.
 * It is rated for a TCD clock of up to 48 MHz.... (and I was able to generate PWM from it without anomalies with
 * it clocked at 128 MHz (32 MHz system clock multiplied by 4, using the 4x multiplier setting that was in the initial
 * io headers, but was pulled from the datasheet before release, and the headers shortly after) - these parts have a
 * ton of headroom on frequency at room temp and under non-adverse conditions)
 */

#if defined(MILLIS_USE_TIMERA0)
  #error "This sketch takes over TCA0, don't use for millis here."
#endif



void setup() {
  // We will be outputting PWM on PA2
  pinMode(PIN_PA2, OUTPUT);
  takeOverTCA0();

  PORTMUX.TCAROUTEA = (PORTMUX.TCAROUTEA & ~(PORTMUX_TCA0_gm)) | PORTMUX_TCA0_PORTA_gc;
  TCA0.SINGLE.CTRLB = (TCA_SINGLE_CMP2EN_bm | TCA_SINGLE_WGMODE_SINGLESLOPE_gc); //Single slope PWM mode, PWM on WO2
  TCA0.SINGLE.PER = 0x00FF; // Count all the way up to 0x00FF (255) - 8-bit PWM
  // At 20MHz, this gives ~78.125kHz PWM
  TCA0.SINGLE.CMP2 = 0;
  TCA0.SINGLE.CTRLA = TCA_SINGLE_ENABLE_bm; //enable the timer with no prescaler
}

void loop() { //Lets generate some output just to prove it works
  static byte pass = 0;
  static unsigned int duty = 255;
  TCA0.SINGLE.CMP2 = duty-- ; //step down the duty cycle each iteration through loop;
  delay(100);  //so we can see the duty cycle changing over time on the scope/with an LED
  if (!duty) {
    if (pass == 0) {
      // After the first pass, lets go up to 100kHz
      pass = 1;
      duty = 199;
      TCA0.SINGLE.PER = 199;
    } else if (pass == 1) {
      //and now the requested 62 kHz (actually 62.11kHz)
      pass = 2;
      duty = 322;
      TCA0.SINGLE.PER = 322;
    } else { // and back to the beginning.
      pass = 0;
      duty = 255;
      TCA0.SINGLE.PER = 255;
    }
  }
}
```

```
/* Example 4: Quick bit of fun with split mode
 * https://github.com/SpenceKonde/DxCore/blob/master/megaavr/extras/TakingOverTCA0.md
 *
 * A quick example of how cool split mode can be - You can get two different PWM frequencies out of the same timer.
 * Split mode only has one mode - both halves of the timer independently count down.
 * Here, we've made it even more interesting by using two frequencies almost identical to each other.... they will
 * "beat" against each other weith a frequency of 1.43 Hz (366 Hz / 256). You should be able to observe that with a
 * bicolor LED (and appropriate resistor) between the two pins. These have two LEDs with opposite polarity, typically
 * a red and a green, connected between two pins... the question is - what will it look like? How will it be different
 * from a single color LED? Make predictions and then test them. When I (Spence) did this, I was wrong.
 */

#if defined(MILLIS_USE_TIMERA0)
  #error "This sketch takes over TCA0, don't use for millis here."
#endif


void setup() {
  // We will be outputting PWM on PD2 amd PD3
  // No need to enable split mode - core has already done that for us.
  pinMode(PIN_PD2, OUTPUT); //PD2 - TCA0 WO2
  pinMode(PIN_PD3, OUTPUT); //PD3 - TCA0 WO3
  PORTMUX.TCAROUTEA = (PORTMUX.TCAROUTEA & ~(PORTMUX_TCA0_gm)) | PORTMUX_TCA0_PORTD_gc; // Variety! Also on all parts!
  TCA0.SPLIT.CTRLB = TCA_SPLIT_LCMP2EN_bm | TCA_SPLIT_HCMP0EN_bm; //PWM on WO2, WO3
  TCA0.SPLIT.LPER = 0xFF; // Count all the way down from 255 on WO0/WO1/WO2
  TCA0.SPLIT.HPER = 0xFE; // Count down from only 254 on WO3/WO4/WO5
  TCA0.SPLIT.LCMP2 = 128; // 50% duty cycle
  TCA0.SPLIT.HCMP0 = 127; // 50% duty cycle
  TCA0.SPLIT.CTRLA = TCA_SPLIT_CLKSEL_DIV256_gc | TCA_SPLIT_ENABLE_bm; //enable the timer with prescaler of 256 - slow it down so the phases shift more slowly, but not so slow it would flicker...
}

void loop() {
  //nothing to do here but enjoy your PWM.
}
```

Result

Examples compiled and uploaded successfully to the board.

Messages

```
Sketch uses 788 bytes (0%) of program storage space. Maximum is 131072 bytes.
Global variables use 8 bytes (0%) of dynamic memory, leaving 16376 bytes for local variables. Maximum is 16384 bytes.

avrdude: Version 6.3-20201216
         Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
         Copyright (c) 2007-2014 Joerg Wunsch

         System wide configuration file is
"C:\Users\ivanFernandez\AppData\Local\Arduino15\packages\Microchip\hardware\megaavr\1.0.0/avrdude.conf"

         Using Port                    : usb
         Using Programmer              : curiosity_updi
avrdude: Found CMSIS-DAP compliant device, using EDBG protocol
         AVR Part                      : AVR128DA48
         Chip Erase delay              : 0 us
         PAGEL                         : P00
         BS2                           : P00
         RESET disposition             : dedicated
         RETRY pulse                   : SCK
         serial program mode           : yes
         parallel program mode         : yes
         Timeout                       : 0
         StabDelay                     : 0
         CmdexeDelay                   : 0
         SyncLoops                     : 0
         ByteDelay                     : 0
         PollIndex                     : 0
         PollValue                     : 0x00
         Memory Detail                 :
```

```
                                  Block Poll               Page                       
Polled
           Memory Type Mode Delay Size  Indx Paged  Size   Size #Pages MinW  MaxW   
ReadBack
           ----------- ---- ----- ----- ---- ------ ------ ---- ------ ----- ----- 
---------
           signature      0     0     0    0 no          3    0      0     0     0 
0x00 0x00
           prodsig        0     0     0    0 no        125  125      0     0     0 
0x00 0x00
           fuses          0     0     0    0 no          9   16      0     0     0 
0x00 0x00
           fuse0          0     0     0    0 no          1    0      0     0     0 
0x00 0x00
           fuse1          0     0     0    0 no          1    0      0     0     0 
0x00 0x00
           fuse2          0     0     0    0 no          1    0      0     0     0 
0x00 0x00
           fuse4          0     0     0    0 no          1    0      0     0     0 
0x00 0x00
           fuse5          0     0     0    0 no          1    0      0     0     0 
0x00 0x00
           fuse6          0     0     0    0 no          1    0      0     0     0 
0x00 0x00
           fuse7          0     0     0    0 no          1    0      0     0     0 
0x00 0x00
           fuse8          0     0     0    0 no          1    0      0     0     0 
0x00 0x00
           lock           0     0     0    0 no          4    1      0     0     0 
0x00 0x00
           data           0     0     0    0 no          0    0      0     0     0 
0x00 0x00
           flash          0     0     0    0 no     131072  512      0     0     0 
0x00 0x00
           eeprom         0     0     0    0 no        512   32      0     0     0 
0x00 0x00

         Programmer Type : JTAGICE3_UPDI
         Description     : Microchip Curiosity in UPDI mode
         ICE hardware version: 0
         ICE firmware version: 1.17 (rel. 514)
         Serial number   : MCHP3280031800001901
         Vtarget         : 3.31 V
         JTAG clock megaAVR/program: 0 kHz
         JTAG clock megaAVR/debug:   0 kHz
         JTAG clock Xmega: 0 kHz
         PDI clock Xmega : 100 kHz

avrdude: Partial Family_ID returned: "    "
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9708 (probably avr128da48)
```

```
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "0b11001001"
avrdude: writing fuse5 (1 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 1 bytes of fuse5 written
avrdude: verifying fuse5 memory against 0b11001001:
avrdude: load data fuse5 data from input file 0b11001001:
avrdude: input file 0b11001001 contains 1 bytes
avrdude: reading on-chip fuse5 data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of fuse5 verified
avrdude: reading input file "0x00"
avrdude: writing fuse7 (1 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 1 bytes of fuse7 written
avrdude: verifying fuse7 memory against 0x00:
avrdude: load data fuse7 data from input file 0x00:
avrdude: input file 0x00 contains 1 bytes
avrdude: reading on-chip fuse7 data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of fuse7 verified
avrdude: reading input file "0x00"
avrdude: writing fuse8 (1 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 1 bytes of fuse8 written
avrdude: verifying fuse8 memory against 0x00:
avrdude: load data fuse8 data from input file 0x00:
avrdude: input file 0x00 contains 1 bytes
avrdude: reading on-chip fuse8 data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of fuse8 verified
avrdude: reading input file
"C:\Users\IVANFE~1\AppData\Local\Temp\arduino_build_547911/TCA0Demo.ino.hex"
avrdude: writing flash (788 bytes):

Writing | ################################################## | 100% 0.31s

```

```
avrdude: 788 bytes of flash written
avrdude: verifying flash memory against
C:\Users\IVANFE~1\AppData\Local\Temp\arduino_build_547911/TCA0Demo.ino.hex:
avrdude: load data flash data from input file
C:\Users\IVANFE~1\AppData\Local\Temp\arduino_build_547911/TCA0Demo.ino.hex:
avrdude: input file
C:\Users\IVANFE~1\AppData\Local\Temp\arduino_build_547911/TCA0Demo.ino.hex
contains 788 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.17s

avrdude: verifying ...
avrdude: 788 bytes of flash verified

avrdude done.  Thank you.
```

Notes

1. Each of the sketches compiled and uploaded successfully to the AVR128DA48 board. This concludes testing of the DXCORE examples within the Team 25 core.