AP32135

TriCore

3-phase complementary PWM with hardware triggered ADC conversion

32bit

Microcontrollers

Never stop thinking

Edition 2011-02

Published by
Infineon Technologies AG
81726 München, Germany

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

AP32135
Revision History: V1.2, 2011-02

Previous Version(s): none

Author: Martin Schrape

Page	Subjects (major changes since last revision)

We Listen to Your Comments
Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

1 Introduction

This application note describes the configuration of a 3-phase complementary Pulse Width Modulation (PWM) [1][2]. Typically, these PWM waveforms drive an H-bridge with high-side and low-side power transistors. To avoid short circuits across this bridge, it is necessary to have dead time between the complementary waveforms. The phase currents are measured and evaluated. Therefore the PWM has to trigger 2 synchronized Analog-to-Digital Converter (ADC) measurements.

This application note has five sections and is primarily written for the TC1796 [3] but can be easily adapted to other AUDO-NG derivatives such as the TC1766 and TC116x-Series. The first section explains how to generate a 3-phase complementary PWM with dead band insertion on the TC1796 using the General Purpose Timer Array (GPTA) module with no CPU overhead. The critical set-up for 0% and 100% duty cycles will be explained in detail.

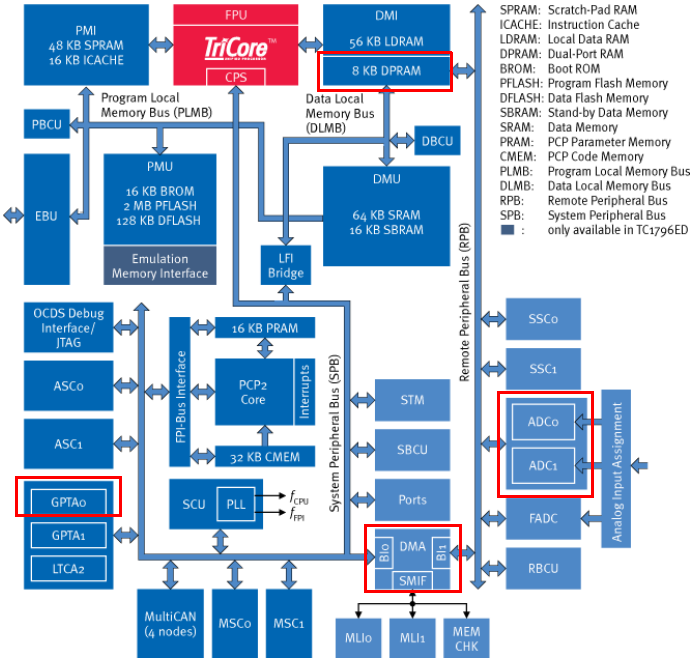

Figure 1 TC1796 block diagram (modules used in this application node are marked in red)

The second section shows how to trigger the ADC module by the GPTA using the External Request Unit (ERU).

The third section details the configuration of the synchronized dual ADC measurement. This section also expands the configuration to a third simultaneous ADC measurement using an interpolation technique.

The fourth section explains how to move the ADC conversion results to the dual ported data memory (DPRAM) using the Direct Memory Access (DMA) controller.

The fifth section explains the calibration that is required to adjust the PWM to the ADC arbiter.

Figure 1 shows the TC1796 block diagram. Modules used in this example are marked red.

The timing diagram in **Figure 2** illustrates the PWM including the dead time, the ADC trigger from the GPTA as well as the ADC channel conversions - sample time marked black -, the DMA transfers, and the TriCore interrupt service routine.

This application note does not cover any aspects of motor control or control algorithms required for such techniques as Space Vector PWM. Such a control algorithm for a 16 kHz PWM requires about 5% of the CPU load for the TC1796, i.e. the TC1796 has enough power and resources to run up to six 3-phase drives.

The configuration is based on Infineon's DAvE **[4]**. Example code is provided for the Tasking TriCore Compiler **[5]**.

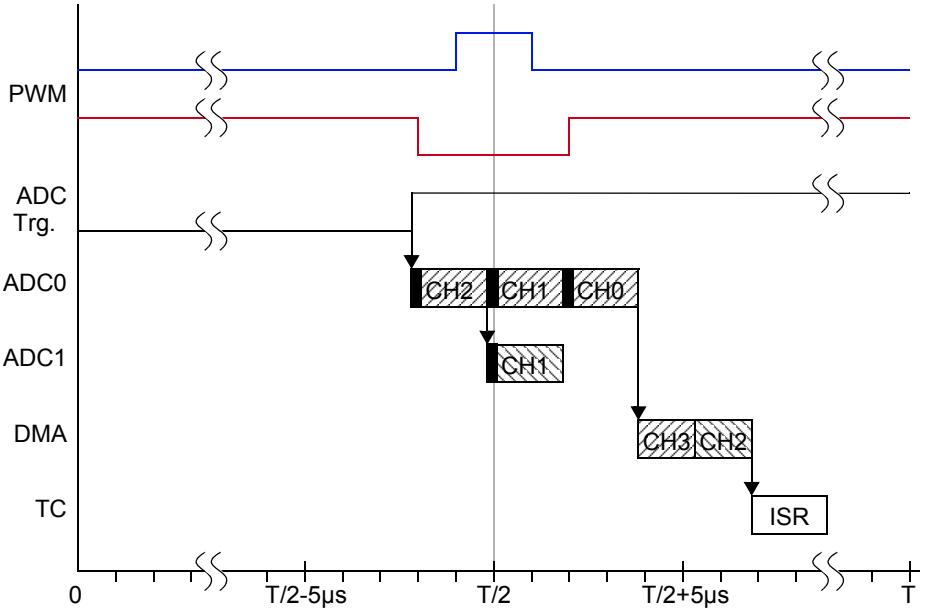

Figure 2 Timing diagram

2 PWM

The GPTA provides a set of timer, compare, and capture functionalities that can be flexibly combined to form signal-measurement and signal-generation units. They are ideal for tasks typical of engine, gearbox, and electrical motor control applications, but can also be used to generate simple and complex signal waveforms needed in other industrial applications.

The TC1796 contains two GPTAs with identical functionality, plus an additional Local Timer Cell Array (LTCA2).

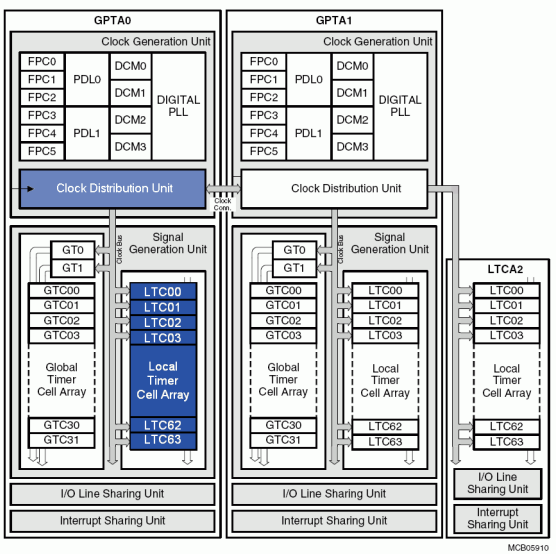

Figure 3 Block diagram of the GPTA modules in the TC1796

GPTA Features

Each of the GPTAs provides a set of hardware modules required for high-speed digital signal processing:

Clock Generation Unit

Filter and Prescaler Cells (FPC) support input noise filtering and prescaler operation.

Phase Discrimination Logic units (PDL) decode the direction information output by a rotation tracking system.

Duty Cycle Measurement Cells (DCM) provide pulse width measurement capabilities.

A digital Phase Locked Loop unit (PLL) generates a programmable number of GPTA module clock ticks during an input signal's period.

Signal Generation Unit

Global Timer units (GT) driven by various clock sources are implemented to operate as a time base for the associated Global Timer Cells (GTC).

GTCs can be programmed to capture the contents of a Global Timer (GT) on an external or internal event. A GTC may also be used to control an external port pin depending on the result of an internal compare operation. GTCs can be logically concatenated to provide a common external port pin with a complex signal waveform.

LTCs operating in Timer, Capture, or Compare Mode may also be logically tied together to drive a common external port pin with a complex signal waveform. LTCs enabled in Timer Mode or Capture Mode can be clocked or triggered by various external or internal events.

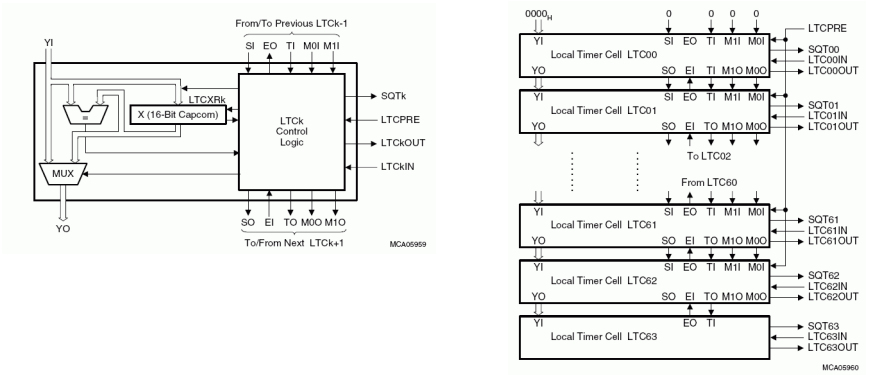

Figure 4 Architecture of LTCs (left) and interconnections between the LTCs (right)

Local Timer Cell (LTC) Functionality

- Local Timer Cell (LTC)
- 64 independent units
- Three basic operating modes (Timer, Capture and Compare) for 63 units
- Special compare modes for one unit
- 16-bit data width
- f_{GPTA} maximum resolution
- $f_{GPTA}/2$ maximum input signal frequency

2.1 Configuration

The PWM configuration uses eight LTCs for each phase (**Figure 5**, **Table 1**). Four additional LTCs are required for the reset timer, the period compare cell, a compare cell to set up the ADC trigger pulse, and a capture cell to measure the actual start of conversion. For 26 cascated LTCs the GPTA module frequency f_{GPTA} needs to be reduced to 37.5 MHz (see 24.6.6 in **[3]**). The reset timer LTC4 uses f_{GPTA} at CLOCK0 as input line operating in level-sensitive mode. The reset timer toggles the select line at every period event, so that the update of a new PWM duty value can be done in a safe way. The software writes the new compare values to the inactive cells and sets the coherent update flag. The hardware toggles the logic state at the end of the period to the new pair of cells. This switching procedure replaces the shadow registers found in other microcontrollers with a much more generic approach.

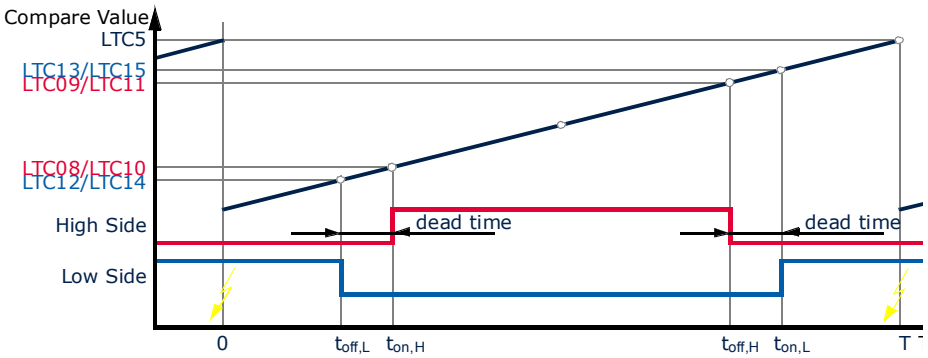

Figure 5 Center-aligned PWM

The high- and low-side PWM configurations are built upon the same building blocks, offering the flexibility of including dead time. Therefore, different on-time and off-time values required by most switching devices, dynamic adjustment and support for adaptive dead time compensation can easily be accomplished.

The configuration of the reset timer is shown in **Figure 6**. The configuration of the period compare cell LTC5 is shown in **Figure 7**. The actual compare register value is set in the user code (File GPTA0.c (Init,3)). A similar configuration is required for LTC6. LTC6 generates the trigger signal for the ADC module on a rising edge. Therefore the output control mode of LTC6 is configured as 'Reset or copy'. The previous cell LTC5 is configured to 'Set' the output. The LTC6 output is connected to GPTA0_OUT3, which is one of three dedicated outputs used to trigger the ADC (**Figure 10**). Additionally, the trigger-gating register TGADC0 must be enabled to connect the ROUT0 line.

Table 1 LTC configuration

LTC			Cell Mode	SL	Output Control Mode	In/Out
4			Reset Timer		Toggle SL on reset	P2.9[1)]
5	PWM Period		Compare	L,H	Set	
6	ADC Trigger		Compare	L,H	Reset or copy	OUT3
7	ADC Trigger Control		Capture	-	Hold	P4.8
8	Phase U	High side	Compare	L	Set	
9			Compare	L	Reset or copy	
10			Compare	H	Set or copy	
11			Compare	H	Reset or copy	P3.0
12		Low side	Compare	L	Reset	
13			Compare	L	Set or copy	
14			Compare	H	Reset or copy	
15			Compare	H	Set or copy	P3.1
16	Phase V	High side	Compare	L	Set	
17			Compare	L	Reset or copy	
18			Compare	H	Set or copy	
19			Compare	H	Reset or copy	P3.8
20		Low side	Compare	L	Reset	
21			Compare	L	Set or copy	
22			Compare	H	Reset or copy	
23			Compare	H	Set or copy	P3.9
24	Phase W	High side	Compare	L	Set	
25			Compare	L	Reset or copy	
26			Compare	H	Set or copy	
27			Compare	H	Reset or copy	P4.0
28		Low side	Compare	L	Reset	
29			Compare	L	Set or copy	
30			Compare	H	Reset or copy	
31			Compare	H	Set or copy	P4.1

1) Only for debugging purposes

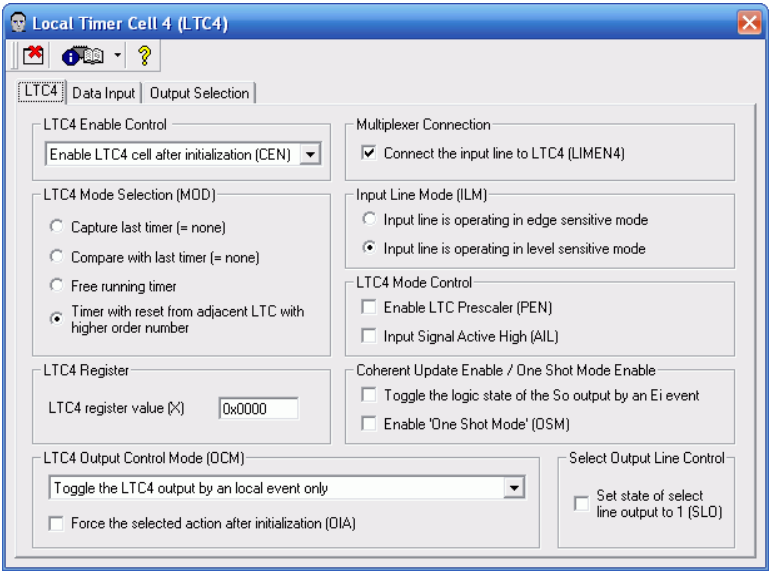

Figure 6 LTC4 Reset timer configuration

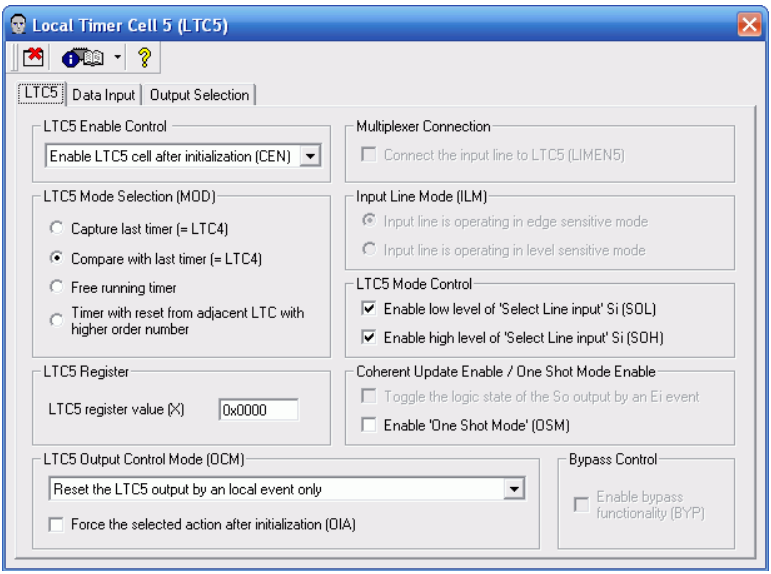

Figure 7 LTC5 Compare cell configuration

The six PWM output signals are controlled by LTC 8 through 31 (LTC8 - LTC31). The configuration is given in **Table 1**. One output signal requires four LTCs. Only two cells are active at the same time, while the others can be reconfigured. The cells' action is forwarded to the last of the four cells and assigned to a port pin.

In a real motor drive application, the update of the duty cycle is the result of the control algorithm. In this example, a sinusoidal modification of the duty cycle is done slowly to demonstrate that the implementation is correct. The basic PWM configuration is defined in the header file MAIN.h (MAIN_Header,3).

```
// USER CODE BEGIN (MAIN_Header,3)
#define LTC_FREQ              37500000  // 37.5MHz
#define PWM_FREQ              16000     // 16.0KHz (62.5us)
#define DEADTIME              1E-6      // 1µs
...
#define DEADTIME_CNTS         (int) (LTC_FREQ * DEADTIME))
#define PWM_PERIOD_CNTS       (int) (LTC_FREQ/PWM_FREQ/20*20-2)
...
// USER CODE END
```

The PWM period is calculated in clock ticks of the reset timer LTC4 as `LTC_FREQ/PWM_FREQ`. The value is rounded to a multiple of the ADC arbitration cycle t_{TIMER} to avoid a jitter between the trigger signal from LTC6 and the ADC conversion start. One ADC arbiter cycle lasts 20 module clock ticks. The module clock is set to the same frequency as LTC4 (the reset timer), so that the arbitration cycle is $t_{TIMER} = 20 \times 1/f_{ADC} = 267$ ns.

The PWM period value has to take into account the fact that the timer resets to -1 and that one clock cycle is required from the compare match value to the reset value (**Figure 8**). Therefore 2 must be subtracted from the PWM period value. Furthermore, the compare values for center-aligned PWMs must be even; this condition is fulfilled due to the rounding. The resulting PWM period is 62.4 µs.

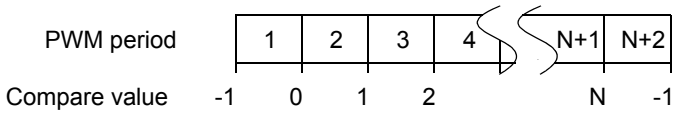

Figure 8 Compare value calculation

An array of scaled sinus values is used for the PWM modification. This array `duty` is initialized at start-up in GPTA0.c (Init,4).

```
// USER CODE BEGIN (Init,4)
for(int i=0;i<sizeof(duty)/sizeof(unsigned);i++)
{
  duty[i] = (PWM_PERIOD_CNTS/2 + 1 + DEADTIME_CNTS) *
            0.5 * (sinf(i * 2 * pi/SINE_STEPS) + 1);
}
// USER CODE END
```

A circular pointer `duty_ptr` is defined in GPTA0.c (GPTA0_General,7) to access the elements in the `duty` array.

```
// USER CODE BEGIN (GPTA0_General,7)
#define SINE_STEPS 360
static __sat unsigned __near __circ duty[SINE_STEPS];
__sat unsigned __near __circ *duty_ptr = duty;
// USER CODE END
```

After the last DMA transaction of ADC results transfered to the DPRAM by the DMA, the DMA's interrupt service routine calls the PWM update function. The calculation of the new LTC values has to take into account the fact that the last LTC determines the action when multiple LTCs in a row that copy the action have the same compare value. At 0% duty cycle, the Set/Reset sequence on the high side results in an inactive signal and Reset determines the output. On the low side, the Reset/Set sequence results in an active signal and Set determines the output. On a period match, the timer resets to -1. For the low side at 100% duty cycle, the compare value of the LTCs with reset action must be saturated to -1 (line 12), so that the output is always inactive (**Figure 9**). For 0% duty, the high side is saturated to the middle of the period value (lines 8 and 10). The Tasking C language extension directly supports the saturation arithmetic of the TriCore instruction set, so that the function **PWM_Update**() executes quickly, avoiding branches. The **PWM_Update**() implementation takes advantage of the address allocation of the LTCs. The step of two compare cell neighbor addresses is 2 words.

```
1  // USER CODE BEGIN (SRN0,1)
2  void PWM_Update(unsigned volatile *ltc_ptr)
3  {
4   __sat unsigned d;
5   for (unsigned i=0;i<3;i++) // for all phases
6   {
7    d = duty_ptr[i*120];
8    *ltc_ptr = PWM_PERIOD_CNTS/2 - (int)(d-DEADTIME_CNTS);
9    ltc_ptr += 2;
10   *ltc_ptr = PWM_PERIOD_CNTS/2 + (int)(d-DEADTIME_CNTS);
11   ltc_ptr += 6;
12   *ltc_ptr=(int)((PWM_PERIOD_CNTS/2+1)-d)-1;
13   ltc_ptr += 2;
14   *ltc_ptr = PWM_PERIOD_CNTS/2 + d;
15   ltc_ptr += 6;
16  }
17 }
// USER CODE END
```

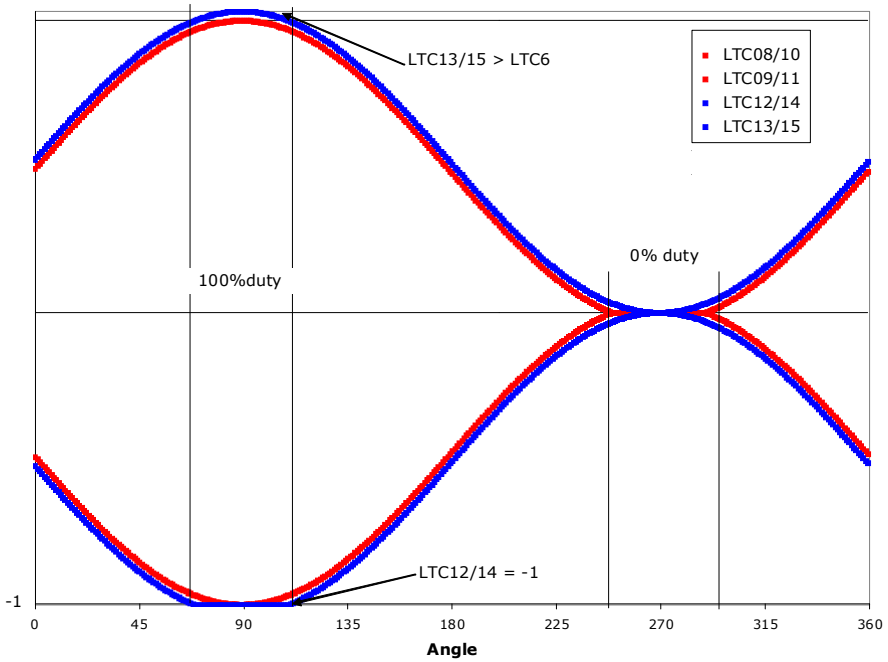

Figure 9 LTC compare values

3 ERU

The External Request Unit (ERU) is part of the System Control Unit (SCU). The GPTA output lines OUT0, OUT8, OUT16, OUT17, OUT24 and OUT25 can be used as input to one of four input channels of the ERU (Figure 5-4 in [3]). Three dedicated output lines of the GPTA0 OUT3, OUT11 and OUT28 are connected to edge-detection logic inside the ERU to obtain a trigger pulse each time a rising edge is detected (Figure 10).

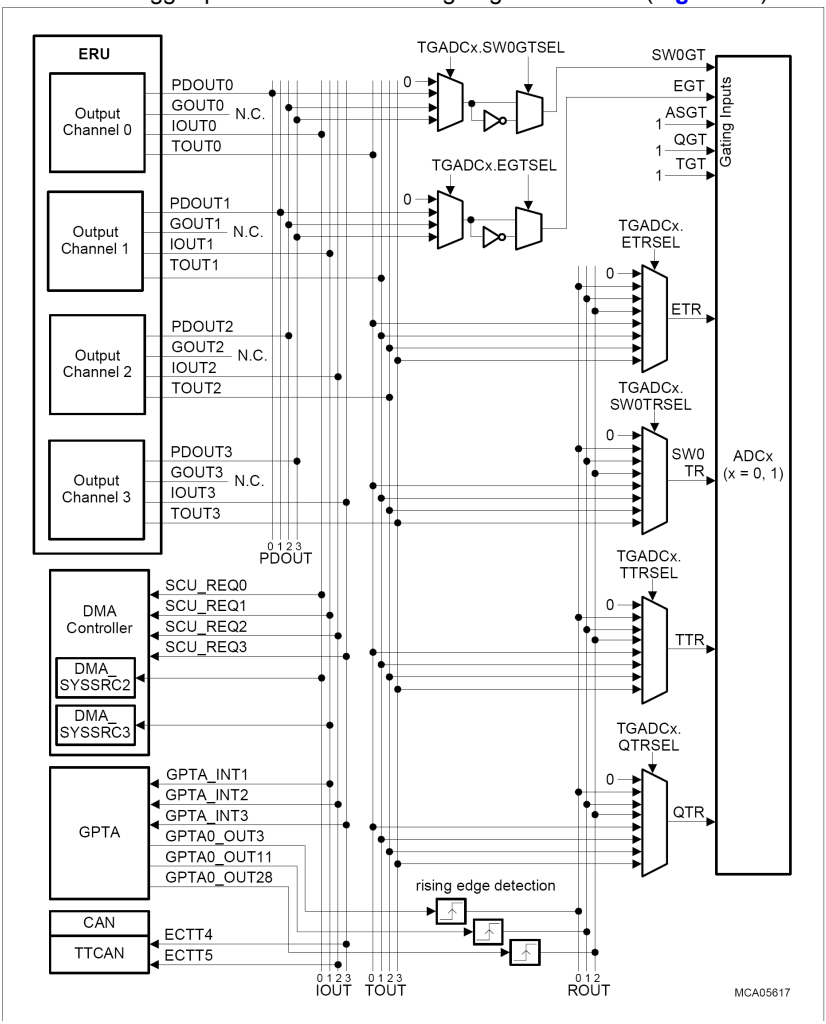

Figure 10 Output Connections of External Request Unit

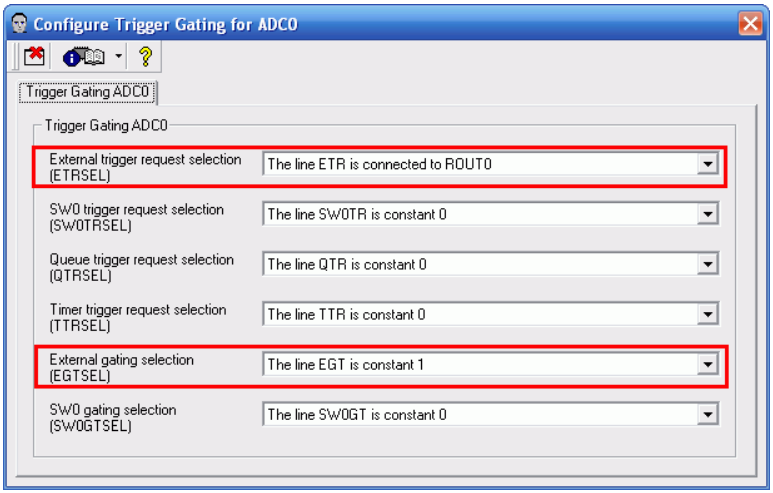

Figure 11 Configure Trigger Gating for ADC0

The configuration requires the external request from the GPTA0_OUT3 to be connected to ROUT0 and to set the external gating to constant 1 (**Figure 11**).

4 ADC

The TC1796 has two ADC modules, ADC0 and ADC1. The trigger signal from the ERU can start a synchronized measurement on both modules. In a drive application two phase currents i_u and i_v will typically be measured. The third one i_w can be calculated by

$$0 = i_u + i_v + i_w \quad (1)$$

Nevertheless, in high-end drive systems, the third phase or the sum i_{sum} will be measured to ensure the integrity of the system. An interpolation technique is used to measure a third current at the same time with two ADC modules . Channel 2 and channel 0 on ADC0 are connected externally to the third input signal. Channel 1 on ADC0 and channel 1 on ADC1 are connected i_u respectively i_v. These channels are synchronized. The external trigger signal from the GPTA0 LTC6 is set as request source to start the conversion of channel 2 to channel 0 on ADC0. All these channels are configured to start conversion on receipt of an external request. The channel with the highest number wins the arbitration (**Figure 12**). The last channel, channel 0, generates a service request when the conversion result is available, which triggers a DMA transfer of the result values to the DPRAM (**Chapter 5**).

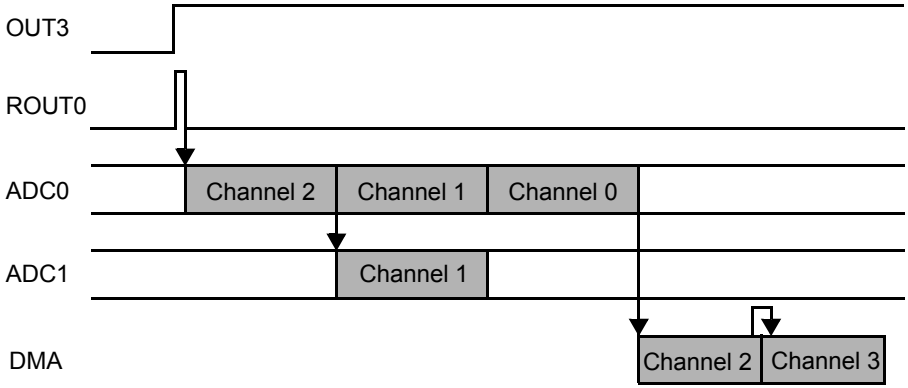

Figure 12 ADC timing

In this example, the ADC module clock is set to f_{ADC} = 75 MHz (t_{ADC} = 13.3 ns). The basic clock f_{BC} must not exceed 40 MHz. Therefore the conversion time control (CTC) value is set to 1, resulting in a basic frequency f_{BC} = 37.5 MHz (t_{BC} = 26.7 ns). The sample time control register CHCONn.STC is set to 0, so that the sample time is t_S = 8 x t_{BC} = 213 ns. All channels are setup to 12-bit resolution. The conversion time is calculated as $t_C = t_S + 56 \times t_{BC} + 2 \times t_{ADC} = t_S + 57 \times t_{BC}$ = 1.73 µs. The time from one channel start to the next channel start of conversion must be a multiple of the arbitration cycle and larger than the conversion time, i.e. 7*t_{TIMER} = 1.87 µs.

In a typical application, LTC6 which controls the GPTA0_OUT3 signal, has to be adjusted so that the midpoint of the sample time of the synchronized ADC channels 1 are at the midpoint of the PWM period (**Figure 2**).

The configurations are shown in **Figure 13** and **Figure 14**.

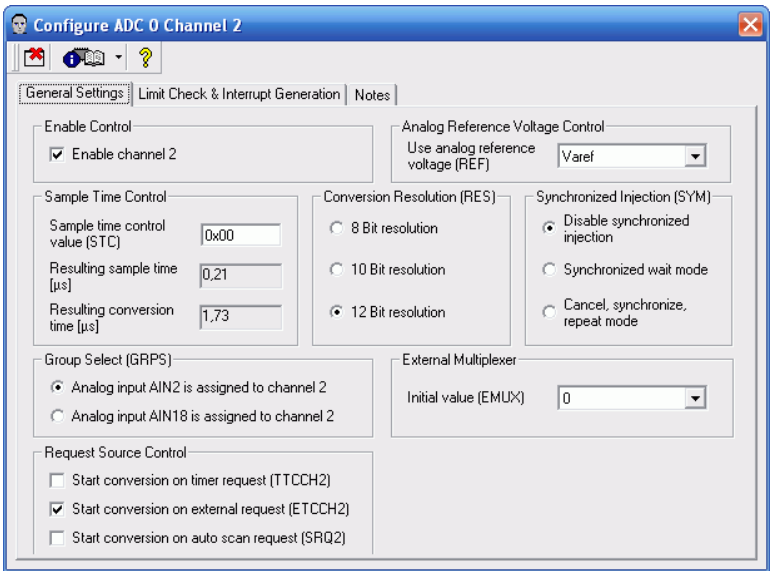

Figure 13 ADC0 Channel 0 and Channel 2 configuration

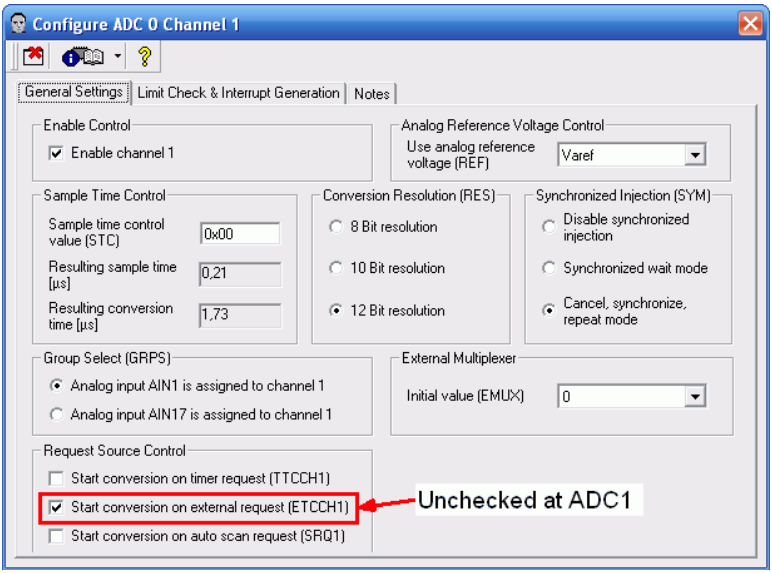

Figure 14 ADC0 Channel 1 and ADC 1 Channel 1 configuration

5 DMA

Two DMA channels are required to transfer the ADC results to the DPRAM. There is one DMA channel for the three results of ADC0 channel 2 to channel 0, and a second for the single result of ADC1 channel 1. Service request line 4 (SR4) of ADC0 connects the ADC request to DMA channel 2. The following line in file ADC0.c (Init,3) sets up the interrupt node pointer of ADC0 channel 0.

```
// USER CODE BEGIN (Init,3)
ADC0_CHCON0 |= 4<<24; // Set interrupt node pointer to SR4
// USER CODE END
```

DMA channel 2 transfers the result of ADC0 channel 2 to channel 0 to the DPRAM. At the end of this transaction, it triggers DMA channel 3 to transfer the ADC1 channel 1 result.

The DMA channel 2 configuration (**Figure 15**) uses one transfer of four half-words to move the lower 16 bits of the registers ADC0_CHSTAT[2:0] to the DPRAM. The fourth move is required by the DMA controller but does not contain useful data in this example. The source address starts at ADC0_CHSTAT0 and is incremented through a 16 byte circular buffer (**Figure 16**). The destination address is the beginning of the DPRAM on the remote peripheral bus (RPB), and the address is incremented through an 8 byte circular buffer. DMA channel 3 is triggered by the end of the channel 2 transfer and transfers only one half-word result from ADC1 channel 1 to the DPRAM. The software defines an array in DMA.c (DMA_General,7) at the beginning of the DPRAM to ease access to the results.

```
// USER CODE BEGIN (DMA_General,7)
unsigned short adc_results[5] __at(0xD000E000);// adc_results[3] unused
// USER CODE END
```

After the completion of the DMA channel 3 transaction, an interrupt is generated to start the control algorithm. Due to the DMA transfer, there is no stall of the CPU core, which would happen if the CPU fetched the results from the peripherals. The control algorithm has a single cycle access to the results. i_w is calculated as the average of the ADC0 channel 0 and channel 2 results.

```
iw = (adc_results[0] + adc_results[2])/2;
```

.

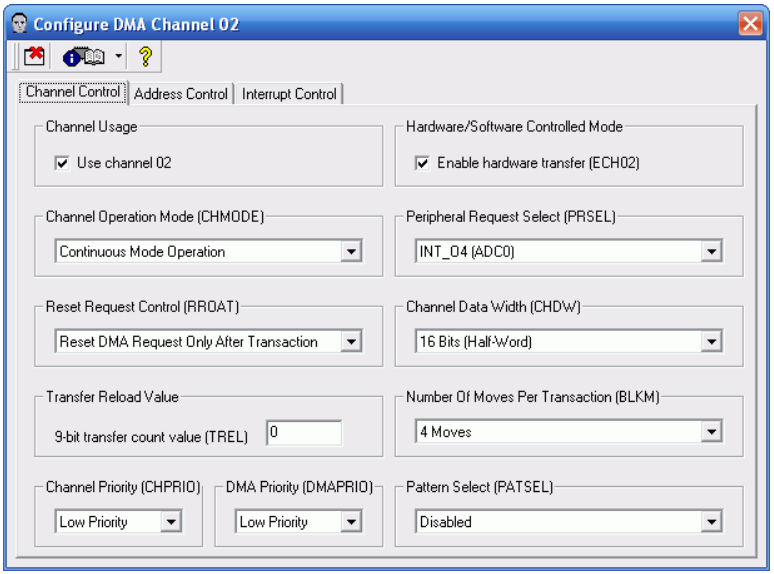

Figure 15 Channel 2 control configuration

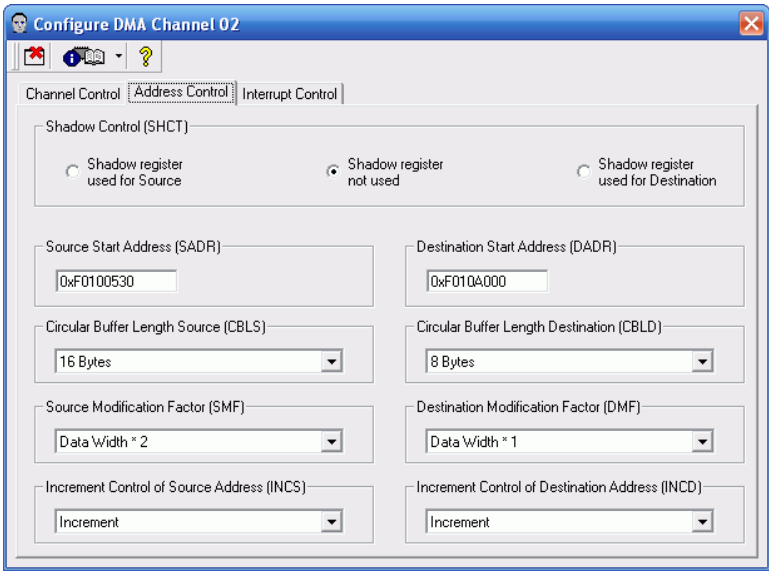

Figure 16 Channel 2 address configuration

6 Calibration

Calibrartion is needed to adjust the ADC trigger from LTC6 to the start-of conversion of ADC0 channel 1. During the calibration, additional port pins are required (**Table 2**).

- The reset timer cell LTC5 is configured to toggle output port pin 2.14.
- GPTA0_OUT3, the ADC trigger signal from LTC6, in connected to port pin 2.11.
- The start of conversion of ADC0 channel 1 can be detected by setting the EMUX0 value to 1, which results in a signal at port pin 7.2. The output is routed to port pin 4.8, the input line of capture cell LTC7.
- A pulse is generated in the DMA interrupt service routine to measure the time of the control algorithm start at port pin 2.8.

Table 2 Logic analyzer setup

Signal	Pin
GPTA LTC04	2.9[1)]
Phase U high	3.0
Phase U low	3.1
Phase V high	3.8
Phase V low	3.9
Phase W high	4.0
Phase W low	4,1
GPTA LTC06	2.11
ADC CH1	7.2 connect to 4.8
TC ISR	2.8[2)]

1) The output of LTC5 is toggled on every timer reset resulting in a PWM trigger frequency of 8 kHz

2) Only used during calibration.

The logic analyzer displays the three complementary phase waveforms (**Figure 17**). Signal **GPTA LTC06** is reset at -1 and set at PWM_PERIOD/2. Signal **ADC CH1** shows the start of the ADC0 channel 1 conversion to the start of the channel 0 conversion. The interval (**Figure 17** Interval B->C) is measured to **1.87 us** as suggested in **Chapter 4**. The conversion time of channel 2 and channel 0 was added manually to **Figure 17**. The interrupt service routine on the TriCore **TC ISR** is issued by the end of the DMA transaction. It contains code to set the port pin P2.8 on entering the routine, and to reset the port on exit. The DMA transfer plus entering the interrupt service routine takes **1.12 µs** (**Figure 17** Interval D->E).

The interval from the LTC6 trigger to the start of ADC channel 1 is **2.12 µs** (**Figure 17** Interval A->B). Adding half of the sample time gives **2.22 µs**, which is about 9 arbitration cycles. A first estimation of the pre-trigger value to LTC6.

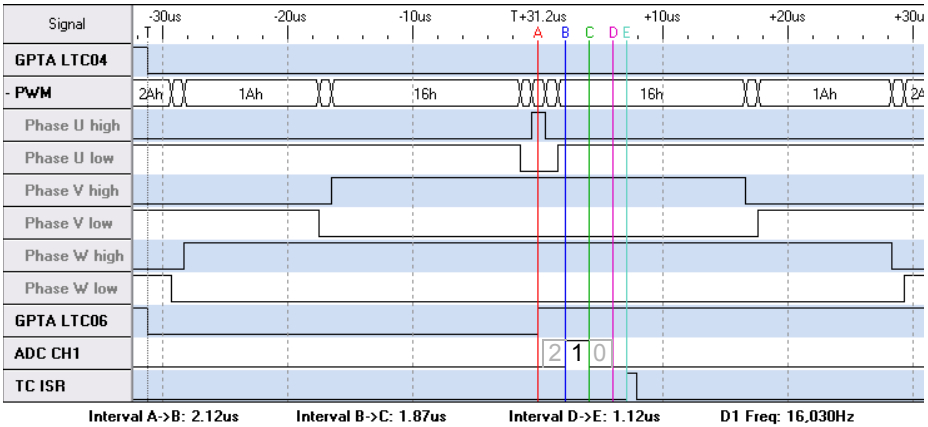

Figure 17 Logic analyzer display before calibration

In a first calibration step, the LTC6 pre-trigger value defined in **MAIN.h** is set to

```
#define ADC_PRETRIGGER_CNTS      (short)(9*20)
```

LTC6 is initialized in **GPTA0.c** as

```
GPTA0_LTCXR06 = PWM_PERIOD_CNTS/2 - ADC_PRETRIGGER_CNTS;
```

Figure 18 displays the result of this calibration step. The ADC0 channel 1 conversion starts 280 ns before the PWM period center.

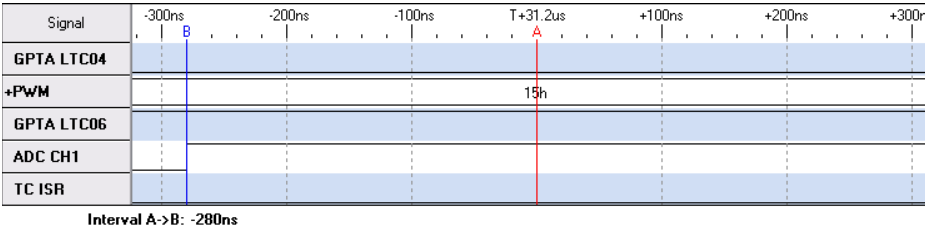

Figure 18 Logic analyzer display after first calibration step

The sample time of channel 1 is t_S = 213 ns. To adjust the midpoint of the sample time period to the PWM trigger, the initial value of the reset timer LTC4 can be used. This approach results in:

```
// USER CODE BEGIN (Init,3)
GPTA0_LTCXR05 = 14;
GPTA0_LTCXR06 = PWM_PERIOD_CNTS;
GPTA0_LTCXR07 = PWM_PERIOD_CNTS/2 - ADC_PRETRIGGER_CNTS;
// USER CODE END
```

Figure 19 shows the results of the second calibration step. The ADC channel 1 starts **105 ns** before the midpoint of the PWM period and is therefore perfectly adjusted to the midpoint of the sample time period.

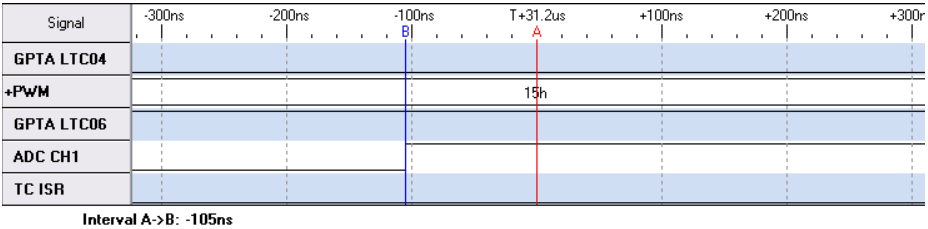

Figure 19 Logic analyzer display after second calibration step

In a highly dynamic system, it is also necessary to adjust the ADC trigger at run-time. The following material describes a run-time calibration. Instead of adjusting the trigger time at start-time, the actual value is captured using LTC7. The captured value is compared and adjusted in an interrupt routine issued by the capture event. The second step of the calibration described above is replaced by reducing the PWM period by one cycle until the set point equals the LTC7 value. **Figure 20** shows the flow diagram of the interrupt routine. It starts with a reset of the service request bit and then stores the global set point variable and the capture value of LTC7 to local variables sp and ltc7. If a modulo 20 operation to the variable sp and variable ltc7 value does not have the same result, then the PWM period is reduced by one cycle and the routine is exited. Otherwise the PWM period will be readjusted to the original value and the difference of sp and ltc7 will be checked. If sp and ltc7 are equal, than the routine disables the service request node and exits. Otherwise the ADC trigger in LTC6 is corrected with the difference of sp and ltc7 and the routine is exited.

In the example code provided with this application note, the interrupt routine is implemented as a channel routine on the Peripheral Control Processor (PCP).

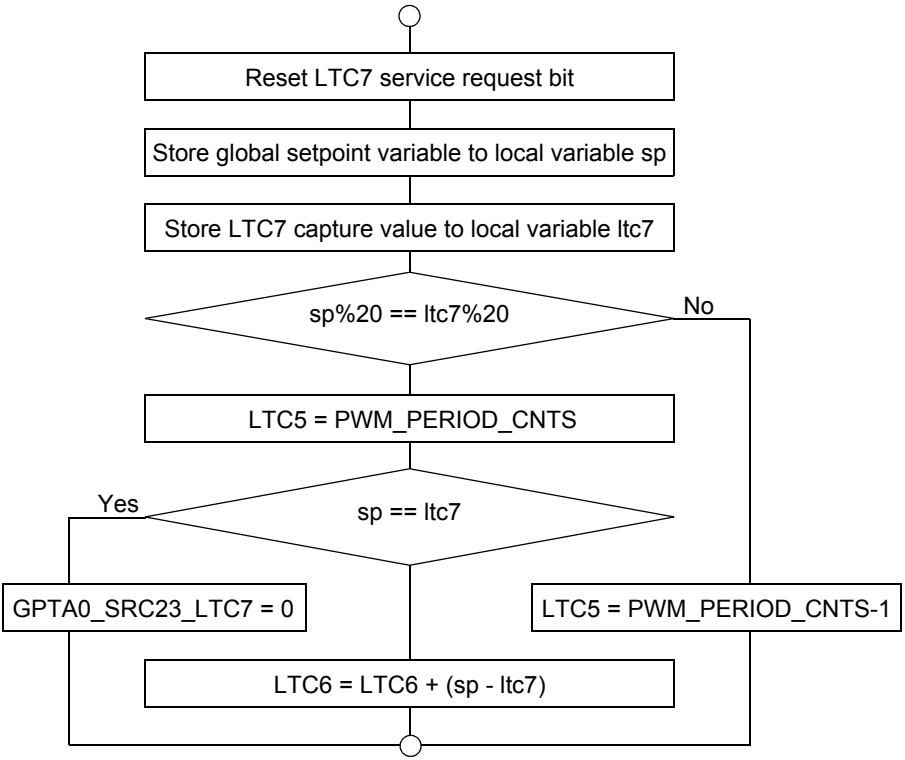

Figure 20 PCP channel program flow diagram

The implementation assumes the PCP is configured for channel-resume mode. Two PCP channel registers, R5 and R6, are initialized with constants. The PCP set point variable PCP_sp is defined as a global variable in the PCP PRAM. It is accessible as an external variable in the TriCore modules through the declaration:

```
extern unsigned _lc_s_PCP_sp;
```

The example program moves the set point stepwise in the main loop to demonstrate the functionality. Bit GPTA0_SRC23_SRE is used as a binary semaphore to signal the TriCore that the set point was reached. The PCP channel program disables the Service Request Node (SRN) when the set point is adjusted. The SRN is enabled again when a new set point is requested.

```
// USER CODE BEGIN (Main,9)
for(;;)
{
  for (_lc_s_PCP_sp = PWM_PERIOD_CNTS/4;
       _lc_s_PCP_sp < PWM_PERIOD_CNTS*3/4;
       _lc_s_PCP_sp += 41)      // Change set point
  {
    GPTA0_SRC23_SRE = 1;    // Use GPTA0_SRC23_SRE as semaphore
    while(GPTA0_SRC23_SRE); // GPTA0_SRC23_SRE disabled by PCPChannel 1
    delay_ms(5000);         // wait 5 sec
  }
}
// USER CODE END
```

The PCP code uses a full context of 8 registers. R7 holds the data pointer DPTR. R6 and R5 are used for constants, so that there are 5 registers free for local variables. No PRAM variable is used except the global set point variable. The program starts with a reset of the service request bit (lines 35-37). It then loads the set point variable to a register and copies the register once. The modulo operation is done in lines 44-48. Taking into account that the set point is a 16-bit variable, the modulo 20 operation can be reduced to only two **dstep** instructions instead of four, therefor reducing the execution time. The LTC7 value is loaded into the register in line 52. The loaded value is also copied. Lines 57-61 calculate the modulo 20 of LTC7. If the results of both modulo operations are not equal (lines 64-65), the PWM period will be reduced (lines 68-70); otherwise the PWM period will be set to the original value (lines 74-75). If the set point equals the LTC7 value, the channel program exits after disabling the service request node (lines 82-85). Otherwise the difference will be added to the ADC trigger at LTC6 (lines 89-94).

```
;*********************************************************************
; PCP PRAM Context Save Area followed by DATA
;*********************************************************************
    .sdecl".pcpdata",DATA, INIT ; declare code section
    .sect ".pcpdata"            ; activate section

    .define PWM_PERIOD_CNTS '4678'

    .space 1*8              ; reserved space for full context areas
ch1_context:
    .word @init_r7(ch1_start,setpoint,0x40);R7
    .word 20;               ; R6 flag that PWM is modified
    .word GPTA0_LTCXR05     ; R5 GPTA0_LTCXR05 address
    .word 0x0               ; R4
    .word 0x0               ; R3
    .word 0x0               ; R2
    .word 0x0               ; R1
    .word 0x0               ; R0

    .global  PCP_sp
PCP_sp:  .type  object
    .size  PCP_sp,1

;*********************************************************************
; PCP CMEM Resume Mode
;*********************************************************************
    .sdecl".pcptext",CODE, INIT    ; declare code section
    .sect ".pcptext"               ; activate section

ch1_exit:
    exit    ec=0,st=0,int=0,ep=1,cc_uc; no interrupt,start @ next PC

ch1_start:
; GPTA0_SRSC2_LTC07 = 1
    ldl.iu  r0, @HI(GPTA0_SRSC2)
    ldl.il  r0, @LO(GPTA0_SRSC2)
    set.f   [r0],7

; r1 = r4 = sp
    ld.pi   r1,[PCP_sp]
    mov     r4,r1,cc_uc

; r3 = r1%20
    rr      r1,8
    rr      r1,8
    dinit   r1,r6
    dstep   r1,r6
```

```
48      dstep   r1,r6
49      mov     r3,r0,cc_uc
50
51  ; r1 = r2 = GPTA0_LTCXR07
52      ld.if   [r5],16
53      mov     r2,r0,cc_uc
54      mov     r1,r0,cc_uc
55
56  ; <r0> = r1%20
57      rr      r1,8
58      rr      r1,8
59      dinit   r1,r6
60      dstep   r1,r6
61      dstep   r1,r6
62
63  ; if (r0 == r3) goto L_1
64      comp    r0,r3, cc_uc
65      jg      L_1,cc_z
66
67  ; GPTA0_LTCXR05 = PWM_PERIOD_CNTS-1; return;
68      ldl.il  r0,PWM_PERIOD_CNTS-1
69      st.f    r0,[r5]
70      jl      ch1_exit
71
72  L_1:
73  ; GPTA0_LTCXR05 = PWM_PERIOD_CNTS
74      ldl.il  r0,PWM_PERIOD_CNTS
75      st.f    r0,[r5]
76
77  ; r4 -= r2; if (!r1) return;
78      sub     r4,r2,cc_uc
79      jg      L_2,cc_nz
80
81  ; GPTA0_SRC23_SRE = 0; return
82      ldl.iu  r0, @HI(GPTA0_SRC23)
83      ldl.il  r0, @LO(GPTA0_SRC23)
84      clr.f   [r0],12
85      jl      ch1_exit
86
87  L_2:
88  ;  if (<r0>=(GPTA0_LTCXR06 + r4))
89      ld.if   [r5],8
90      add     r0,r4,cc_uc
91
92  ;  GPTA0_LTCXR06 = r0
93      st.if   [r5],8
94      jl       ch1_exit
```

7 Implementation

The example is based on a configuration done with DAvE and some additional lines of code added manually. The following is a list of the DAvE dialogs in which changes have been made to set up the example:

- File > Project Settings > General
- File > Project Settings > System Clock
- File > Project Settings > Interrupt System
- File > Project Settings > PCP System
- GPTA_CLOCK > Module Clock
- GPTA_CLOCK > Timer Clock Control
- GPTA0 > Input Pins > IN32 > GPTA0_IN32
- GPTA0 > Local Timer > LTC4 > LTC4
- GPTA0 > Local Timer > LTC4 > Data Input
- GPTA0 > Local Timer > LTC4 > Output Selection
- GPTA0 > Local Timer > LTC5 > LTC5
- GPTA0 > Local Timer > LTC6 > LTC6
- GPTA0 > Local Timer > LTC6 > Output Selection
- GPTA0 > Local Timer > LTC7 > LTC7
- GPTA0 > Local Timer > LTC7 > Data Input
- GPTA0 > Local Timer > LTC8 > LTC8
- GPTA0 > Local Timer > LTC9 > LTC9
- GPTA0 > Local Timer > LTC10 > LTC10
- GPTA0 > Local Timer > LTC11 > LTC11
- GPTA0 > Local Timer > LTC11 > Output Selection
- GPTA0 > Local Timer > LTC12 > LTC12
- GPTA0 > Local Timer > LTC13 > LTC13
- GPTA0 > Local Timer > LTC14 > LTC14
- GPTA0 > Local Timer > LTC15 > LTC15
- GPTA0 > Local Timer > LTC15 > Output Selection
- GPTA0 > Local Timer > LTC16 > LTC16
- GPTA0 > Local Timer > LTC17 > LTC17
- GPTA0 > Local Timer > LTC18 > LTC18
- GPTA0 > Local Timer > LTC19 > LTC19
- GPTA0 > Local Timer > LTC19 > Output Selection
- GPTA0 > Local Timer > LTC20 > LTC20
- GPTA0 > Local Timer > LTC21 > LTC21
- GPTA0 > Local Timer > LTC22 > LTC22
- GPTA0 > Local Timer > LTC23 > LTC23
- GPTA0 > Local Timer > LTC23 > Output Selection
- GPTA0 > Local Timer > LTC24 > LTC24
- GPTA0 > Local Timer > LTC25 > LTC25
- GPTA0 > Local Timer > LTC26 > LTC26

- GPTA0 > Local Timer > LTC27 > LTC27
- GPTA0 > Local Timer > LTC27 > Output Selection
- GPTA0 > Local Timer > LTC28 > LTC28
- GPTA0 > Local Timer > LTC29 > LTC29
- GPTA0 > Local Timer > LTC30 > LTC30
- GPTA0 > Local Timer > LTC31 > LTC31
- GPTA0 > Local Timer > LTC31 > Output Selection
- GPTA0 > Local Timer > LTC31 > LTC31
- GPTA0 > Local Timer > LTC31 > LTC31
- GPTA0 > Output Pins > OUT3 >GPTA0_OUT3
- GPTA0 > Output Pins > OUT6 >GPTA0_OUT6
- GPTA0 > Output Pins > OUT8 >GPTA0_OUT8
- GPTA0 > Output Pins > OUT9 >GPTA0_OUT9
- GPTA0 > Output Pins > OUT16 >GPTA0_OUT16
- GPTA0 > Output Pins > OUT17 >GPTA0_OUT17
- GPTA0 > Output Pins > OUT24 >GPTA0_OUT24
- GPTA0 > Output Pins > OUT25 >GPTA0_OUT25
- GPTA0 > Functions
- GPTA0 > SRN > Service Request Node 22-23 (LTC0-7) >SRN
- GPTA0 > SRN > Service Request Node 22-23 (LTC0-7) >Interrupts
- SCU > Watchdog
- SCU > ERU > Trigger Gating ADC0 > Trigger Gating ADC0
- SCU > Functions
- ADC Clock > Module Clock
- ADC0 > General > AD0EMUX0;AD0EMUX0;AD0EMUX0 > AD0EMUX0
- ADC0 > Channels > Configure Channel 2 > General Settings
- ADC0 > Channels > Configure Channel 1 > General Settings
- ADC0 > Channels > Configure Channel 0 > General Settings
- ADC0 > Channels > Configure Channel 0 > Limit Check & Interrupt Generation
- ADC0 > SRN
- ADC0 > Functions
- ADC1 > Channels > Configure Channel 1 > General Settings
- ADC1 > Functions
- DMA > Block 0 > DMA Channel 02 > Channel Control
- DMA > Block 0 > DMA Channel 02 > Address Control
- DMA > Block 0 > DMA Channel 03 > Channel Control
- DMA > Block 0 > DMA Channel 03 > Address Control
- DMA > Block 0 > DMA Channel 03 > Interrupt Control
- DMA > Memory 0
- DMA > SRN
- DMA > Interrupts
- DMA > Functions

The following is a list contains of sections in which source code was added to the code generated by DAvE:

- MAIN.h (MAIN_Header,3)
- MAIN.c (MAIN_General,6)
- MAIN.c (Main,9)
- ADC0.c (Init,3)
- DMA.c (DMA_General,6)
- DMA.c (DMA_General,7)
- DMA.c (SRN0,1)
- DMA.c (SRN0,2)
- DMA.c (SRN0,603)
- GPTA0.c (GPTA0_General,2)
- GPTA0.c (GPTA0_General,6)
- GPTA0.c (GPTA0_General,7)
- GPTA0.c (Init,3)
- GPTA0.c (Init,4)
- gpta.pcp

8 References

[1] Application Note, AUDO1: Compare Unit implementation with GPTA and PCPs

[2] Application Note AP32084, TriCore Sinusodial 3-Phase Output Generation Using The TriCore General Purpose Timer Array

[3] TC1796 User's Manual V2.0, July 2007

[4] http://www.infineon.com/dave

[5] http://www.tasking.com/tricore

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG