Never stop thinking

TriCore

AP32169

Electric motor control

Application Note

V1.0 2011-09

Microcontrollers

Edition 2011-09

Published by
Infineon Technologies AG
81726 Munich, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices, please contact the 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 the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only 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.

TC1782

Revision History: V1.0, 2011-09

Previous Version: V0.2D1, 2010-09

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Is there any information in this document that you feel is wrong, unclear or missing?
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

Table of Contents

1 Preface

This application note describes the implementation of a motion controller on the TriCore [1] AUDO MAX-family. It explains the configuration of 3-phase complementary Pulse Width Modulation (PWM). The document is aimed at developers who write or design real-time motion control applications on the TriCore. This document looks specifically at those features of the TriCore architecture that make it an attractive platform for real-time embedded systems, focusing particularly on the peripheral modules: The General Purpose Timer Array (GPTA), the Direct Memory Access (DMA) Module and the Analog to Digital Converter (ADC) but also pointing out the advantages of the CPU core to run the control algorithm.

This application note assumes that readers have access to the TriCore Architecture Manual [2] and the TC1782 Users Manual [3], and have at least some general knowledge of the TriCore instruction set, the architectural features and peripheral modules. The application notes explaining the principles of a single 3-phase PWM setup using the GPTA [4,5] and Field Oriented control [6] are particularly pertinent to potential readers of this document.

See References on page 22 for more information on the TriCore and other relevant documentation. It is assumed that most readers will be generally familiar with the features and functions of motion control systems.

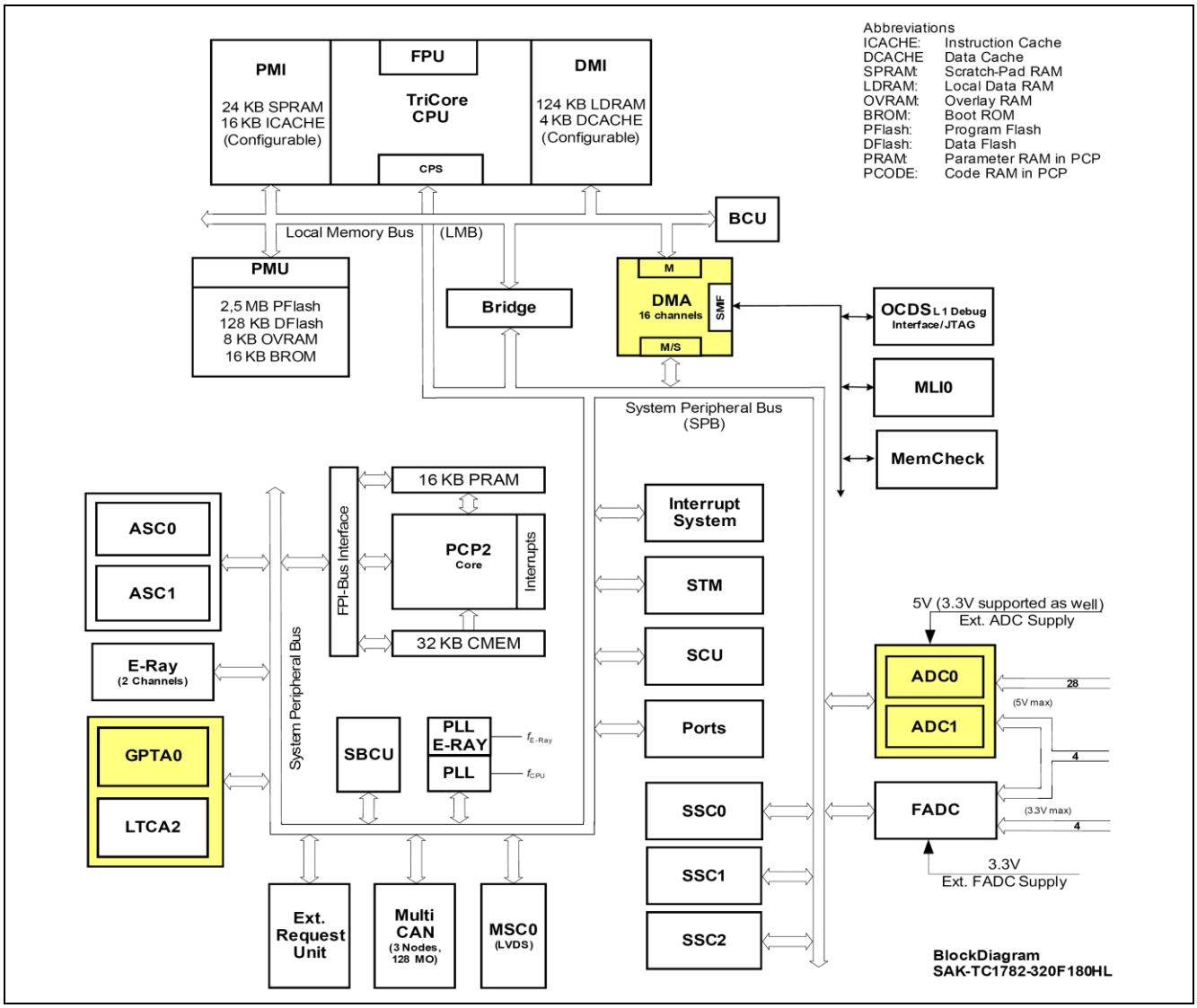

Figure 1 TC1782 Block Diagram

2 Introduction

Figure 1 shows the TC1782 block diagram. Modules used in this application note are marked yellow. This section 2 gives an introduction to the principles of generating 3-phase complementary PWM signals on the TriCore. Section 3 explains an efficient configuration and initialization of the GPTA, ADC and DMA module. Section 4 illustrates the example application that is provided with this application note.

Typically, the 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 insert a dead time between the complementary waveforms. The three phase currents are measured simultaneously and synchronized to the PWM signal output. The TC1782 has two ADC kernels, each with 16 channels, so that the simultaneous acquisition of more than two analog signals is only possible by interpolation of two symmetric measuring points before and after the trigger time.

The GPTA is set up to generate the PWM signal output and the trigger signal for the ADC. The timing diagram in Figure 2 illustrates the complementary PWM outputs with only one complementary output signal pair (high-side and low-side) of the 3-phase PWM shown. A dead time between the switching on and off of the high- and low-side switches avoids a short on the power devices. The timer unit also issues a request signal to trigger the ADC so that the acquisition of input signal 2 on channel 2 is done at the period start. Multiple input signals are measured twice:

- Input signal 0 on ADC0 is measured twice: with channel 0 and on channel 4.
- Input signal 1 on ADC0 is measured twice: with channel 1 and on channel 3.
- Input signal 1 on ADC1 is measured twice: with channel 1 and on channel 3.

In each case the two results are summed up to obtain an interpolated value for the period start time.

At the end of the last ADC conversion two DMA channels move the ADC results into the TriCore local data memory (LDRAM). The last DMA transfer triggers a TriCore interrupt which executes the control algorithm.

The scan of eleven ADC conversions at 12-bit resolution requires about 10 µs, the two DMA transfers about 4 µs and the interrupt including the FOC algorithm takes up about 2 µs. The PWM update for the next period is finished in less than 10 µs after the start of the PWM period. The CPU load is only caused by the control algorithm and reaches about 2% in total.

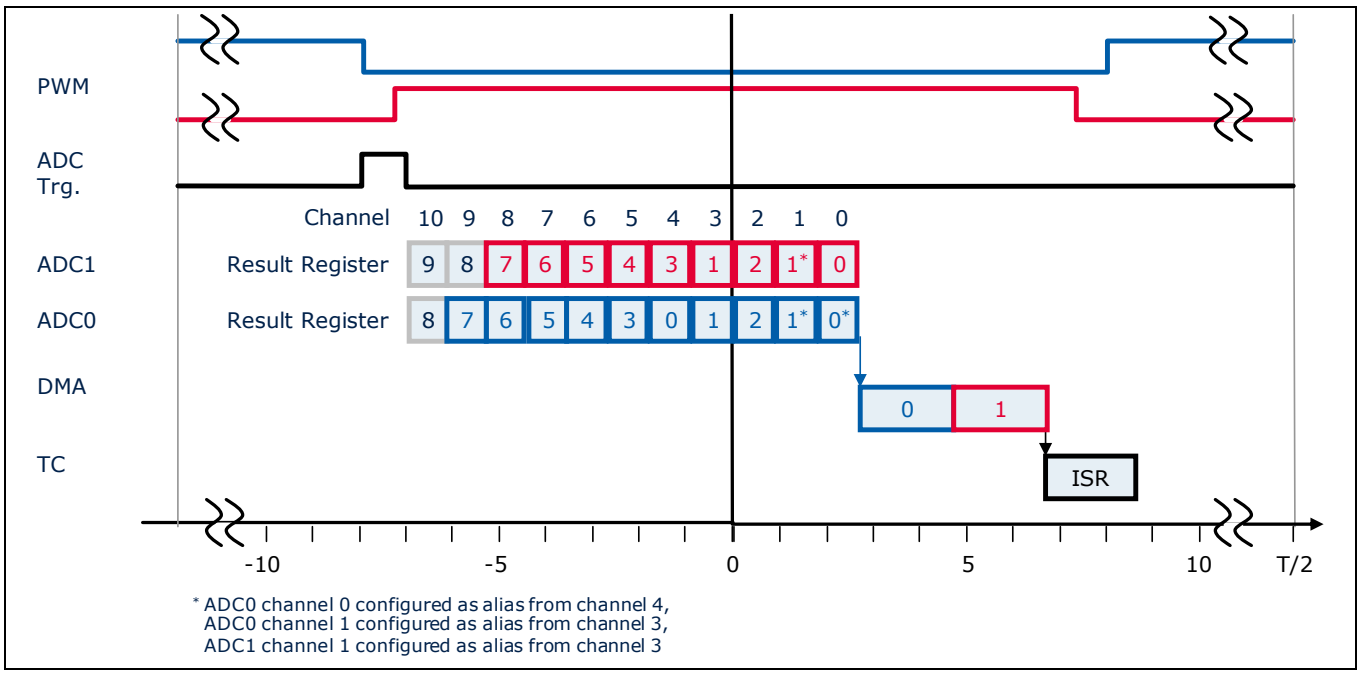

Figure 2 Timing Diagram

3 Configuration

3.1 PWM

The TC1782 contains one General Purpose Timer Arrays (GPTA0), plus an additional Local Timer Cell Array (LTCA2) (see Figure 3).

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 optimized for tasks typical for engine, gearbox, and electrical motor control applications, but can also be used to generate simple and complex signal waveforms required for other industrial applications.

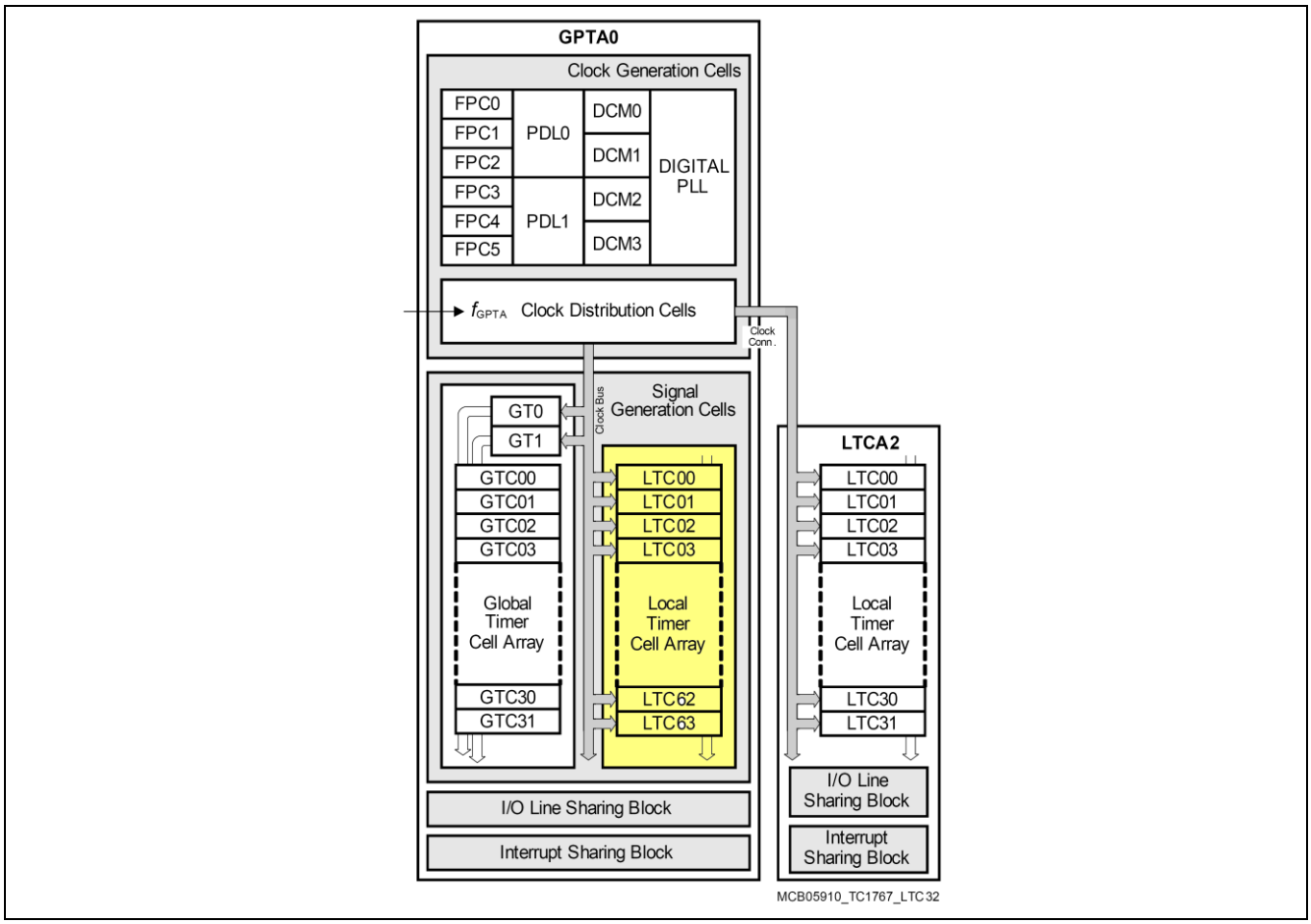

Figure 3 General Block Diagram of the GPTA Modules

This application note uses the LTC array of the GPTA0 to generate the 3-phase complementary PWM signals. The configuration is shown in Table 1 and Figure 4. A 3-phase PWM requires 26 LTCs (LTC4-5 and LTC8-31). Two additional LTCs (LTC6-7) are used to generate the ADC trigger signal. The LTC output is routed through the multiplexer to the ports and the trigger multiplexer (Figure 5). The output port pins P2.1 and P2.0 are used for debug purpose during development.

Table 1 GPTA0 configuration

				GPTA		
LTC	**Mode**	**Output Multiplexer Group**	**I/O group**		**Output**	**Port**
4	Timer					
5	Period	OMG10	IOG0		(OUT33)	(P2.1)
6	Compare			**ADC**		
7	Compare	OMG10	OTMG0 and IOG0		(OUT32), TRIG03	(P2.0)
8-10	Compare			**PWM**		
11	Compare	OMG11	IOG1		OUT8	P0.8
12-14	Compare					
15	Compare	OMG11	IOG1		OUT9	P0.9
16-18	Compare					
19	Compare	OMG12	IOG2		OUT18	P1.2
20-22	Compare					
23	Compare	OMG12	IOG2		OUT19	P1.3
24-26	Compare					
27	Compare	OMG13	IOG3		OUT26	P1.10
28-30	Compare					
31	Compare	OMG13	IOG3		OUT27	P1.11

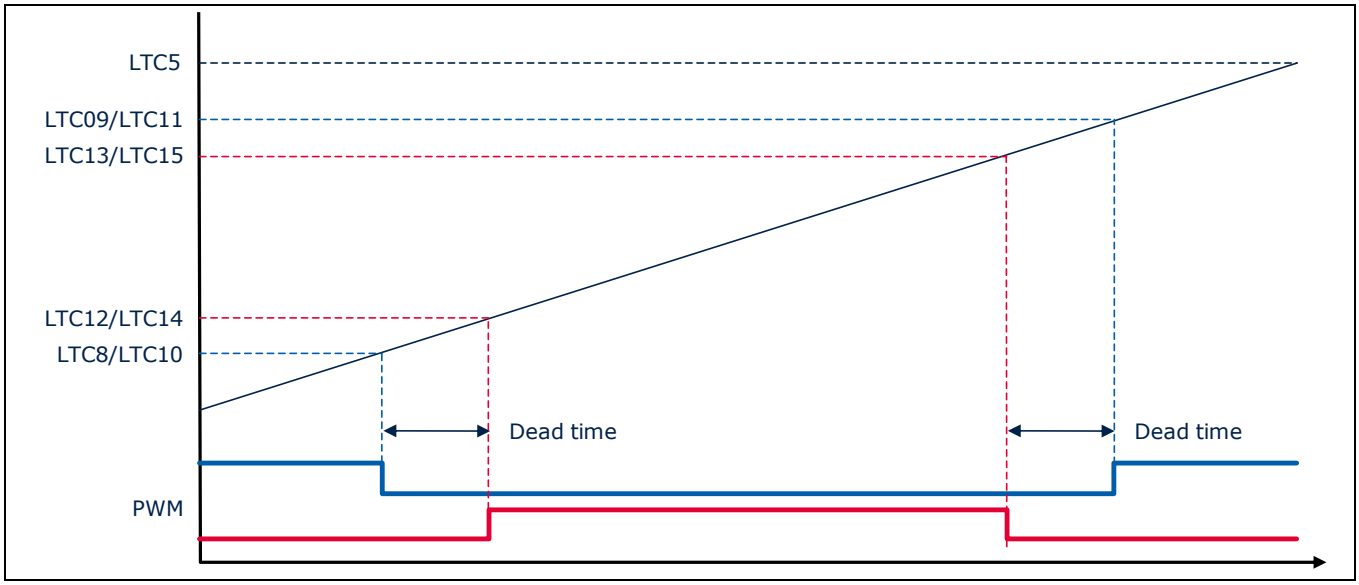

Figure 4 GPTA0 configuration

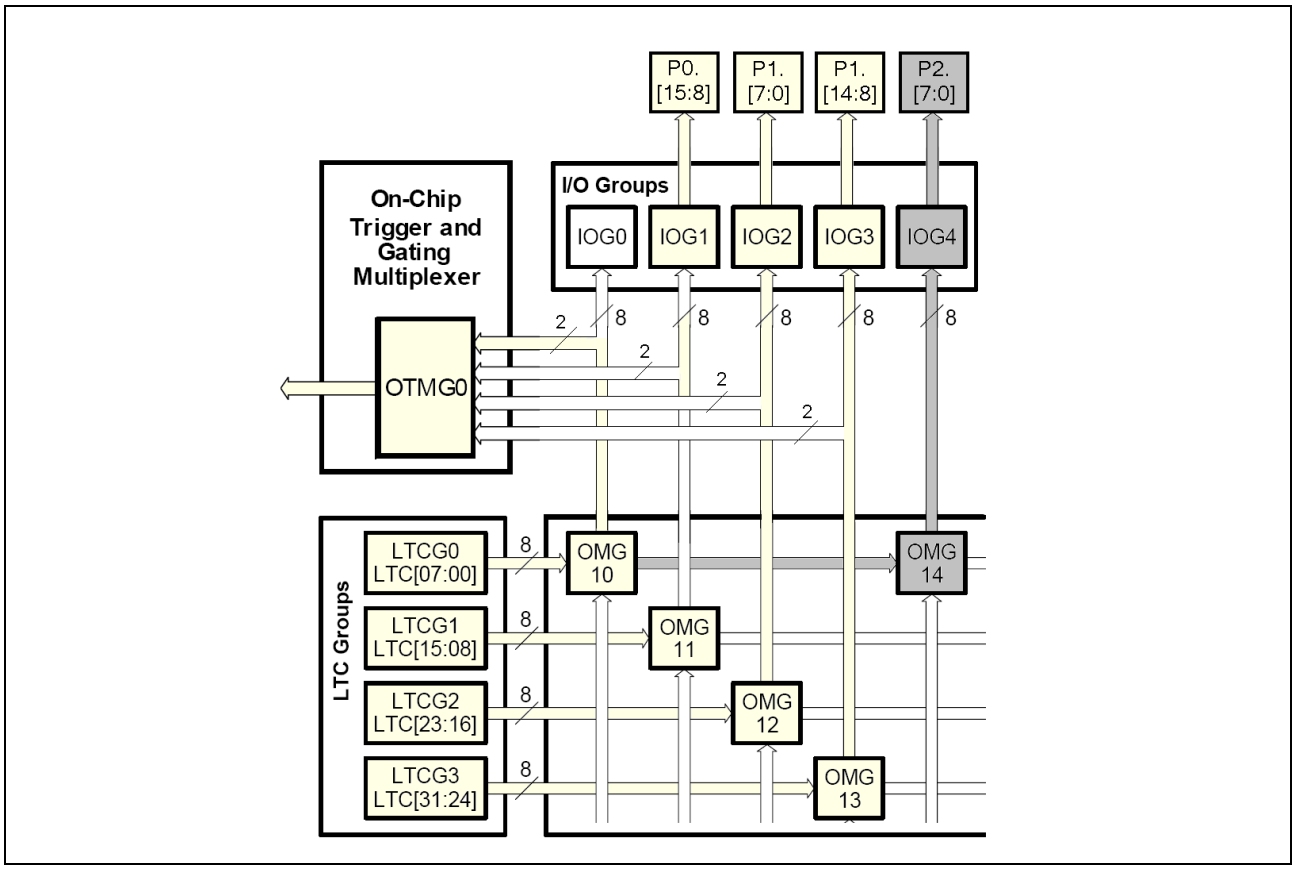

Figure 5 Details of the Output Multiplexer (see also [3] Figure 21-66, 21-69, 21-95)

The PWM initialization sequence is listed in Listing 1. The GPTA register address map shows that pairs of LTC control registers and LTC X registers are located sequentially in memory, so that register addressing of the LTCs from LTC4 to LTC31 can simply be done by incrementing a pointer. A local variable is initialized with the address of the LTC4 control register (Line 219). Line 220/221 configures the control register of the reset timer LTC4. The next local timer cell LTC5 is initialized as compare cell (Line 226) with a compare value of the PWM period (Line 229).

The ADC trigger signal generated by LTC6 and LTC7 is configured by Line 230-235. The ADC will be triggered by a falling edge before the PWM period. The trigger time is adjustable at a multiple of the ADC conversion time value.

```
37 #define ADC_CONVERSION_CNTS (2+(4+0+12)*5) // (2*tADC+(4+STC+n)*tADCI)/tLTC
```

with tADC = 1/90MHz, tLTC = 90MHz, tADCI = 5/90MHz

The clock configuration is done in cstart.c. The major part of the the configuration is utilzed for the multiplexer (Line 252-317 and Figure 5). Each byte which is writen to the Multiplexer Register Array Data In Register MRADIN is related to one GPTA output. Three bits of the lower nibble determines one out of eight input lines. Two bits of the higher nibble determine the output group: A “1” selects the LTC groups LTCG0 to LTCG3, a “2”would selects the LTC groups LTCG4 to LTCG7.

Note: Line 286 for example writes the OMCRL3 value to the multiplexer FIFO array. It is the FIFO element 30 (see [4] Figure 22-28) which determines OUT24 to OUT27. The OUT27 byte is initialized with 0x17 selecting input 7 of LTCG3, i.e. LTC31.

```
215   /*
216    * General Purpose Timer Array (GPTA)
217    */
218
219   uint32_t volatile * ltc_ptr = &GPTA0_LTCCTR04.U;
220   *ltc_ptr++ = 0x0103; // Timer,
221   *ltc_ptr++ = 0;
222
223 #ifdef DEBUG
224   *ltc_ptr++ = 0x0831; // Compare, SOL/SOH active, Toggle output
225 #else
226   *ltc_ptr++ = 0x0031; // Compare, SOL/SOH active
227 #endif
228
229   *ltc_ptr++ = 2 * PWM_PERIOD_CENTER_CNTS;
230   *ltc_ptr++ = 0x1C31; // Compare, SOL/SOH active, Set output
231   *ltc_ptr++ = 2 * PWM_PERIOD_CENTER_CNTS - ADC_CONVERSIONS_BEFORE_PWM_START
232         * ADC_CONVERSION_CNTS - LTC_FREQ * 1e-6;
233   *ltc_ptr++ = 0x3431; // Compare, SOL/SOH active, Reset or copy output
234   *ltc_ptr++ = 2 * PWM_PERIOD_CENTER_CNTS - ADC_CONVERSIONS_BEFORE_PWM_START
235         * ADC_CONVERSION_CNTS;
236
237   uint64_t *pp = (uint64_t *) ltc_ptr;
238   for (int32_t i = 0; i < 3; i++)
239   {
240     // High side
241     *(uint32_t*) pp++ = 0x1811; // Compare, SOL active, Set output
242     *(uint32_t*) pp++ = 0x3011; // Compare, SOL active, Reset or copy output
243     *(uint32_t*) pp++ = 0x3821; // Compare, SOH active, Set or copy output
244     *(uint32_t*) pp++ = 0x3021; // Compare, SOH active, Reset or copy output
245     // Low side
246     *(uint32_t*) pp++ = 0x1011; // Compare, SOL active, Reset output
247     *(uint32_t*) pp++ = 0x3811; // Compare, SOL active, Set or copy output
248     *(uint32_t*) pp++ = 0x3021; // Compare, SOH active, Reset or copy output
249     *(uint32_t*) pp++ = 0x3821; // Compare, SOH active, Set or copy output
250   }
251
252   GPTA0_MRACTL.B.MAEN = 0; // disable multiplexer array
253   while (GPTA0_MRACTL.B.MAEN != 0)
254     ; // wait for bit MAEN
255   GPTA0_MRACTL.B.WCRES = 1; // reset count
256   GPTA0_MRADIN.U = 0; // 53 GPTA0_OTMCR1
257   GPTA0_MRADIN.U = 0; // 52 GPTA0_OTMCR0 {,,,,OUT0_TRIG03,,,OUTx_TRIG00}
258   GPTA0_MRADIN.U = 0; // 51 GPTA0_OMCRH13
259   GPTA0_MRADIN.U = 0; // 50 GPTA0_OMCRL13
260   GPTA0_MRADIN.U = 0; // 49 GPTA0_OMCRH12
261   GPTA0_MRADIN.U = 0; // 48 GPTA0_OMCRL12
262   GPTA0_MRADIN.U = 0; // 47 GPTA0_OMCRH11
263   GPTA0_MRADIN.U = 0; // 46 GPTA0_OMCRL11
264   GPTA0_MRADIN.U = 0; // 45 GPTA0_OMCRH10
265   GPTA0_MRADIN.U = 0; // 44 GPTA0_OMCRL10
266   GPTA0_MRADIN.U = 0; // 43 GPTA0_OMCRH9
267   GPTA0_MRADIN.U = 0; // 42 GPTA0_OMCRL9
268   GPTA0_MRADIN.U = 0; // 41 GPTA0_OMCRH8
269   GPTA0_MRADIN.U = 0; // 40 GPTA0_OMCRL8
270   GPTA0_MRADIN.U = 0; // 39 GPTA0_OMCRH7
271   GPTA0_MRADIN.U = 0; // 38 GPTA0_OMCRL7
272   GPTA0_MRADIN.U = 0; // 37 GPTA0_OMCRH6
273   GPTA0_MRADIN.U = 0; // 36 GPTA0_OMCRL6
274   GPTA0_MRADIN.U = 0; // 35 GPTA0_OMCRH5
275   GPTA0_MRADIN.U = 0; // 34 GPTA0_OMCRL5
276   GPTA0_MRADIN.U = 0; // 33 GPTA0_OMCRH4
277 #ifdef DEBUG
278
279   GPTA0_MRADIN.U = 0x00001517;// 32 GPTA0_OMCRL4 {,,LTC05_OUT33,LTC07_OUT32}
280 #else
281
282   GPTA0_MRADIN.U = 0; // 32 GPTA0_OMCRL4
```

```
283 #endif
284
285   GPTA0_MRADIN.U = 0; // 31 GPTA0_OMCRH3
286   GPTA0_MRADIN.U = 0x17130000; // 30 GPTA0_OMCRL3 {LTC31_OUT27,LTC27_OUT26,,}
287   GPTA0_MRADIN.U = 0; // 29 GPTA0_OMCRH2
288   GPTA0_MRADIN.U = 0x17130000; // 28 GPTA0_OMCRL2 {LTC23_OUT19,LTC19_OUT18,,}
289   GPTA0_MRADIN.U = 0; // 27 GPTA0_OMCRH1
290   GPTA0_MRADIN.U = 0x00001713; // 26 GPTA0_OMCRL1 {,,LTC15_OUT9,LTC11_OUT8}
291   GPTA0_MRADIN.U = 0; // 25 GPTA0_OMCRH0
292   GPTA0_MRADIN.U = 0x00000017; // 24 GPTA0_OMCRL0 {,,,LTC07_OUT0}
293   GPTA0_MRADIN.U = 0; // 23 GPTA0_LIMCRH7
294   GPTA0_MRADIN.U = 0; // 22 GPTA0_LIMCRL7
295   GPTA0_MRADIN.U = 0; // 21 GPTA0_LIMCRH6
296   GPTA0_MRADIN.U = 0; // 20 GPTA0_LIMCRL6
297   GPTA0_MRADIN.U = 0; // 19 GPTA0_LIMCRH5
298   GPTA0_MRADIN.U = 0; // 18 GPTA0_LIMCRL5
299   GPTA0_MRADIN.U = 0; // 17 GPTA0_LIMCRH4
300   GPTA0_MRADIN.U = 0; // 16 GPTA0_LIMCRL4
301   GPTA0_MRADIN.U = 0; // 15 GPTA0_LIMCRH3
302   GPTA0_MRADIN.U = 0; // 14 GPTA0_LIMCRL3
303   GPTA0_MRADIN.U = 0; // 13 GPTA0_LIMCRH2
304   GPTA0_MRADIN.U = 0; // 12 GPTA0_LIMCRL2
305   GPTA0_MRADIN.U = 0; // 11 GPTA0_LIMCRH1
306   GPTA0_MRADIN.U = 0; // 10 GPTA0_LIMCRL1
307   GPTA0_MRADIN.U = 0x000000B0; // 9 GPTA0_LIMCRH0 {0,0,0,CLK0_LTC04}
308   GPTA0_MRADIN.U = 0; // 8 GPTA0_LIMCRL0
309   GPTA0_MRADIN.U = 0; // 7 GPTA0_GIMCRH3
310   GPTA0_MRADIN.U = 0; // 6 GPTA0_GIMCRL3
311   GPTA0_MRADIN.U = 0; // 5 GPTA0_GIMCRH2
312   GPTA0_MRADIN.U = 0; // 4 GPTA0_GIMCRL2
313   GPTA0_MRADIN.U = 0; // 3 GPTA0_GIMCRH1
314   GPTA0_MRADIN.U = 0; // 2 GPTA0_GIMCRL1
315   GPTA0_MRADIN.U = 0; // 1 GPTA0_GIMCRH0
316   GPTA0_MRADIN.U = 0; // 0 GPTA0_GIMCRL0
317   GPTA0_MRACTL.B.MAEN = 1; // enable multiplexer array
318
319   GPTA0_EDCTR.U = 0x100; // Enable GPTA0 timer clock
```

Listing 1 GPTA0 Initialization

3.2 ADC

All ADC input signals in the system are measured every PWM period on a scan request for channel 10 to 0 on both ADC modules. Table 2 shows a sampling schema with a total of nineteen input signals: IU, IV, IW, Res_A, Res_B, U5, U6, U7, U8, U9, T10, U16, U20, U21, U22, U23, U24, T25, T26. Input signal T10, T25 and T26 are used to control input signals autonously in the backgroud. They are configured with a limit checking feature. If the configured boundary is exceeded a channel event is raised.

The two ADC kernels are synchronized so that the phase currents IU and IV are measured at the same time. Other signals which should also be measured at the same time as IU and IV, can be measured twice, before and after the phase currents IU and IV, so that a linear interpolation gives the value at the sample time of IU and IV.

The scan is arranged in a way that the signals which are critical for the PWM control algorithm are measured last so that the delay from the last measurement to the PWM interrupt routine control is minimized.

The phase current IW uses the alias feature which is available for channel 0 and 1. The programmed alias channel number is replacing the internally requested number for analog input multiplexer of the converter. The internally requested channel number is taken into account for all other internal actions and the synchronization request. IW on ADC0 is measured by channel 4 and channel 0, but channel 0 is configured as alias of channel 4. I.e. both use the same settings including the port pin AN4. In addition channel 0 is configured to use the same result register as channel 4, so that in one scan the values are written twice to result register 0. These two values are summed up by a data reduction filter. On ADC1 kernel this configuration leads to two conversions of the input signal at An19.

ADC1 CH0 is not triggered by any master channel on ADC0.

This configuration results in a total of 2 × 8 values - the content of the result register RESR0 to RESR7 of ADC0 and ADC1 – that needs to be transferred to the TriCore per PWM period.

Table 2 ADC0/ADC1 configuration

ADC 0				ADC 1			
Pin	**Channel**	**Signal**	**Result Register**	**Pin**	**Channel**	**Signal**	**Result Register**
AN10	CH10	T10	RESR8	AN26	CH10	T26	RESR9
AN9	CH9	U9	RESR7	AN25	CH9	T25	RESR8
AN8	CH8	U8	RESR6	AN24	CH8	U24	RESR7
AN7	CH7	U7	RESR5	AN23	CH7	U23	RESR6
AN6	CH6	U6	RESR4	AN22	CH6	U22	RESR5
AN5	CH5	U5	RESR3	AN21	CH5	U21	RESR4
AN4	CH4	IW	RESR0	AN20	CH4	U20	RESR3
AN3	CH3	Res_A	RESR1	AN19	CH3	Res_B	RESR1
AN2	CH2	IU	RESR2	AN18	CH2	IV	RESR2
AN3	CH1	Res_A	RESR1	AN19	CH1	Res_B	RESR1
AN4	CH0	IW	RESR0	AN16	CH0	U16	RESR0

Listing 2 shows the complete ADC initialization. Module clock configuration, permanent arbitration settings, and start of power up calibration is done in cstart.c. The period t_{ARB} of an arbitration round is given by:

t_{ARB} = GLOBCTR.ARBRND × (GLOBCTR.DIVD + 1) / f_{ADC}

add configured to

t_{ARB} = 4 × (0 + 1) / 90 MHZ = 44 ns.

Note: The arbitration of the ADC module in the TC1782 is much faster then in the TC1796. The timing calibration which was required due to the minimum arbitration round of 267ns in the TC1796 described in AP32135 is no longer necessary.

One input class in each ADC is configured to a 12-bit resolution (Line 375-376). Eleven channels in each ADC are configured by the channel control register ADCn_CHCTRx (Line 379-401). Each channel uses the result register (Bit Field RESRSEL) as defined in Table 2 and the input class 0 (Bit Field ICLSEL). ADC0 control registers are set to a potential synchronization master by the SYNC bit 7. Limit check on boundaries is configured in ADC0_CHCTR10, ADC1_CHCTR10 and ADC1_CHCTR9. The Synchronization Control Register (SYNCTR) sets ADC0 to a master, ADC1 to a slaves (Line 404-405).

The alias register is configured so that ADC0 channel 0 is an alias of channel 3 (Line 408-409).

The data reduction filter is set up for all register which should accumulate two conversions (Line 412-417).

ADC0 is configured for external trigger events (Line 420) from a falling edge on GPTA_TRIG03 (Line 421, Figure 6). The conversion request is set-up using the scan request source 1 for channel 10 to 0 (Line 422). The initialization is completed by enabling the arbitration for the request source 1 (Line 423), a wait for the calibration (Line 431-434) which was started in cstart.c and request to the master to switch on the analog part (Line 436). The slave will be switched on by the master.

```
355   /*
356    * Analog to Digital Converter (ADC)
357    *
372    */
373
374   // Input classes
375   ADC0_INPCR0.U = 0x0100; //  12bit input class 0
376   ADC1_INPCR0.U = 0x0100;
377
378   // Channel configure
379   ADC0_CHCTR10.U = 0x807B; // Use result register 8, limit check boundary:
    upper 3, lower 2, generate interrupt if in area III
380   ADC0_CHCTR9.U = 0x7000; // Use result register 7, enable sync request
381   ADC0_CHCTR8.U = 0x6080; // Use result register 6, enable sync request
382   ADC0_CHCTR7.U = 0x5080; // Use result register 5, enable sync request
383   ADC0_CHCTR6.U = 0x4080; // Use result register 4, enable sync request
384   ADC0_CHCTR5.U = 0x3080; // Use result register 3, enable sync request
385   ADC0_CHCTR4.U = 0x0080; // Use result register 0, enable sync request
386   ADC0_CHCTR3.U = 0x1080; // Use result register 1, enable sync request
387   ADC0_CHCTR2.U = 0x2080; // Use result register 2, enable sync request
388   ADC0_CHCTR1.U = 0x1080; // Use result register 1, enable sync request
389   ADC0_CHCTR0.U = 0x0080; // Use result register 0, enable sync request
390
391   ADC1_CHCTR10.U = 0x907B; // Use result register 8, limit check boundary:
    upper 3, lower 2, generate interrupt if in area III
392   ADC1_CHCTR9.U = 0x807B; // Use result register 8, limit check boundary:
    upper 3, lower 2, generate interrupt if in area III
393   ADC1_CHCTR8.U = 0x7000; // Use result register 7
394   ADC1_CHCTR7.U = 0x6000; // Use result register 6
395   ADC1_CHCTR6.U = 0x5000; // Use result register 5
396   ADC1_CHCTR5.U = 0x4000; // Use result register 4
397   ADC1_CHCTR4.U = 0x3000; // Use result register 3
398   ADC1_CHCTR3.U = 0x1000; // Use result register 1
399   ADC1_CHCTR2.U = 0x2000; // Use result register 2
400   ADC1_CHCTR1.U = 0x1000; // Use result register 1
401   ADC1_CHCTR0.U = 0x0000; // Use result register 0
402
403   // Synchronization
404   ADC0_SYNCTR.U = 0x10; // Evaluate Ready Input R1. Kernel is a sync master
405   ADC1_SYNCTR.U = 0x11; // Evaluate Ready Input R1. Kernel is a sync slave
406
407   // Alias
408   ADC0_ALR0.U = 0x304; // Use AN3 as input to ADC0:CH1. Use AN4 as input to
    ADC0:CH0.
409   ADC1_ALR0.U = 0x300; // Use AN19 as input to ADC1:CH1.
410
411   // Data reduction
412   ADC0_RCR0.U = 0x1; // add 2 conversions in ADC0:CH0
413   ADC0_RCR1.U = 0x1; // add 2 conversions in ADC0:CH1
414   ADC0_RCR1.U = 0x1; // add 2 conversions in ADC1:CH1
```

```
415   ADC0_RCR10.U = 0x3; // add 4 conversions in ADC0:CH10
416   ADC0_RCR9.U = 0x3; // add 4 conversions in ADC0:CH9
417   ADC1_RCR10.U = 0x3; // add 4 conversions in ADC1:CH10
418
419   // Scan
420   ADC0_CRMR1.U = 0xD; // Gate always enabled, external trigger, enable source
    interrupt
421   ADC0_RSIR1.U = 0x1000; // Trigger on falling edge of GPTA_TRIG03
422   ADC0_CRCR1.U = 0x7FF; // Scan channel 10 to 0
423   ADC0_ASENR.U = 1 << 1; // Enable Arbitration Slot 1
424
425   // Limit Check Boundary
426   ADC1_LCBR3.U = 0xFFF * 0.95; // 95% full range
427   ADC1_LCBR2.U = 0xFFF * 0.9; // 90% full range
428
429   ADC0_SRC0.U = 0x1000 | TEMP_INT;
430
431   while (ADC0_GLOBSTR.B.CAL)
432     ; // wait for calibration finished
433   while (ADC1_GLOBSTR.B.CAL)
434     ; // wait for calibration finished
435   // Switch on analog part. Only switch master.
436   ADC0_GLOBCTR.U |= 0x00000300; // Analog part switch on
```

Listing 2 ADC Initialization

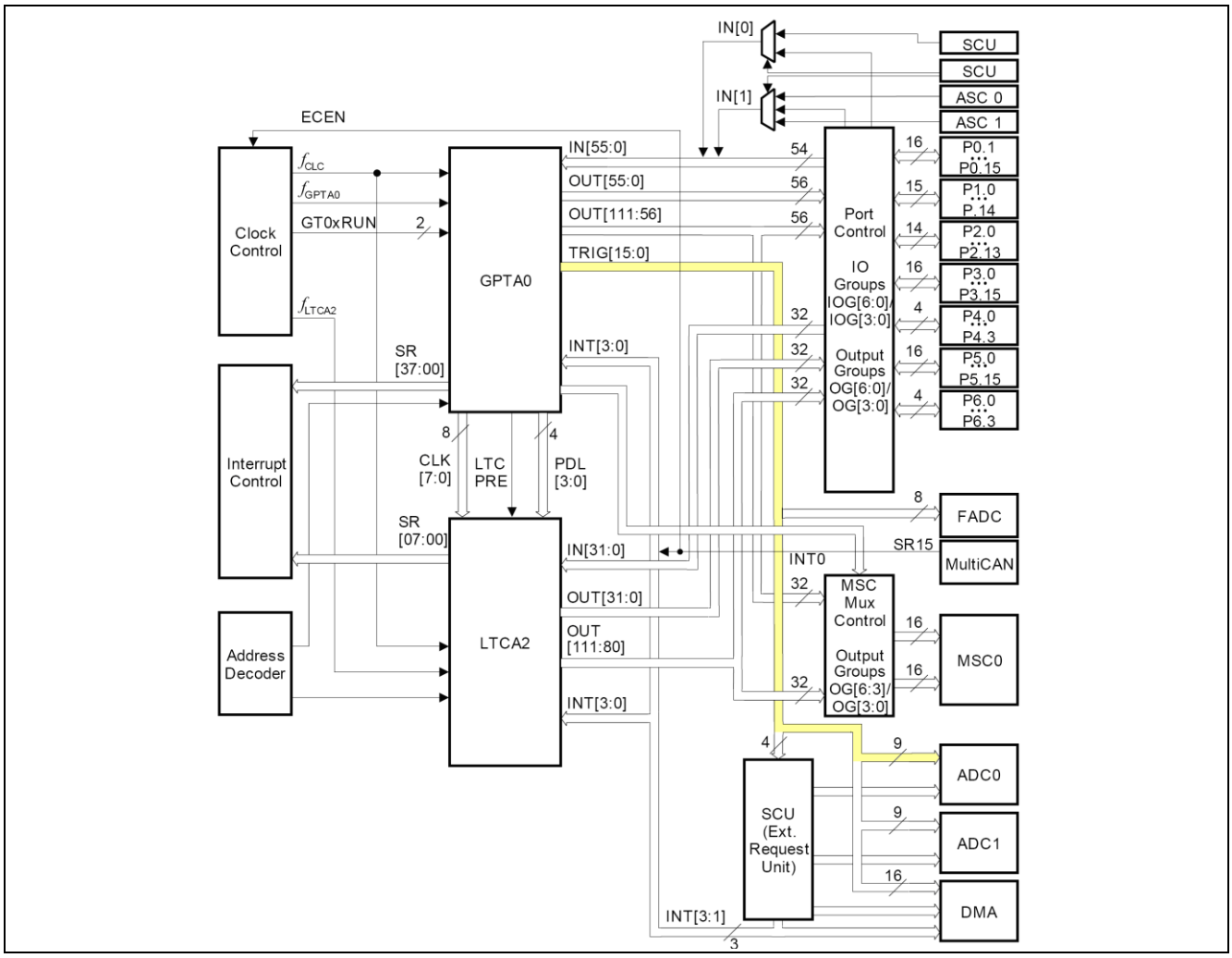

Figure 6 Block Diagram of GPTA Implementation. Request line from GPTA Trigger and to ADC0 maked yellow.

3.3 DMA

Listing 3 shows the complete DMA. The initialization of the access ranges is done in cstart.c. DMA channel 0 transfers the eight result register of ADC0 (Line 442). DMA channel 1 transfers eight result register of ADC1 (Line 448). DMA channel 0 is triggered by the hardware transaction request ADC_SR00 from the last ADC conversion of the scan (Line 442, 454 and Figure 7). The channel is configured with a 16 Byte circular source buffer starting at ADC0_RESR0 (Line 443). The Circular Buffer Length Source (CBLS) bit field of the Address Control Register determines the number of address bits that are updated. I.e. with CBLS equal to 4 SADR[31:4] is not updated (see Table 3).

The eight values are transferred to the destination address in the DMI RAM adc_results[0][0] to adc_results[7][0]. The adc_result array is organized so that the phase currents i_u, i_v, i_w of the motor are located next to each other in the memory so that a single double word access by LD.D can get all three 16-bit values. The DMA channel 0 triggers DMA channel 1 using the DMA internal request note pointer SR09 (Line 448, 454 and Figure 7). DMA channel 1 transfers the eight results of ADC1 to adc_results[0][1] to adc_results[7][1] and issues an interrupt on the TriCore (Line 456).

```
438   /*
439    * Direct Memory Access (DMA)
440    */
441
442   DMA_CHCR00.U = 0xD03B3000; // 16bit data width, triggered by ADC_SR00, 1
    transfers of 8 move
443   DMA_SADR00.U = (uint32_t) & ADC0_RESR0.U; // source address
444   DMA_DADR00.U = (uint32_t) & adc_results[0][0]; // destination address
445   DMA_ADRCR00.U = 0x5599; // 32Byte circular buffer, increment 4Byte
446   DMA_CHICR00.U = 9 << 8 | 2 << 2; // DMA internal request, note pointer SR09
447
448   DMA_CHCR01.U = 0xD03B0000; // 16bit data width, triggered by DMA_SR09, 1
    transfers of 8 move
449   DMA_SADR01.U = (uint32_t) & ADC1_RESR0.U; // source address
450   DMA_DADR01.U = (uint32_t) & adc_results[0][1]; // destination address
451   DMA_ADRCR01.U = 0x5599; // 32Byte circular buffer, increment 4Byte
452   DMA_CHICR01.U = 0 << 8 | 2 << 2; // Interrupt enable, note pointer SR00
453
454   DMA_HTREQ.U = 0x3; // DMA Hardware Transaction REQuest for channel 0 and 1
455
456   DMA_SRC0.U = 0x1000 | PWM_INT;
```

Listing 3 DMA Initialization

Table 3 Source circular buffer
ADC 0 (Base Address $F010\ 1000_H$)/ADC1(Base Address $F010\ 1400_H$)

Register	Offset	Binary Address (grey not update, white updated)
RESR0	180_H	$1\ 1000\ 0000_B$
RESR1	184_H	$1\ 1000\ 0100_B$
RESR2	188_H	$1\ 1000\ 1000_B$
RESR3	$18C_H$	$1\ 1000\ 1100_B$
RESR4	190_H	$1\ 1001\ 0000_B$
RESR5	188_H	$1\ 1001\ 0100_B$
RESR6	188_H	$1\ 1001\ 1000_B$
RESR7	$18C_H$	$1\ 1001\ 1100_B$

Table 4 Destination circular buffer for ADC0 results

Source	Binary Address
adc_results[0][0]	XXXX XXXX XXXX XXXX XXXX XXXX XXX0 0000$_B$
adc_results[1][0]	XXXX XXXX XXXX XXXX XXXX XXXX XXX0 0100$_B$
adc_results[2][0]	XXXX XXXX XXXX XXXX XXXX XXXX XXX0 1000$_B$
adc_results[3][0]	XXXX XXXX XXXX XXXX XXXX XXXX XXX0 1100$_B$
adc_results[4][0]	XXXX XXXX XXXX XXXX XXXX XXXX XXX1 0000$_B$
adc_results[5][0]	XXXX XXXX XXXX XXXX XXXX XXXX XXX1 0100$_B$
adc_results[6][0]	XXXX XXXX XXXX XXXX XXXX XXXX XXX1 1000$_B$
adc_results[7][0]	XXXX XXXX XXXX XXXX XXXX XXXX XXX1 1100$_B$

Table 5 Destination circular buffer for ADC1 results

Source	Binary Address
adc_results[0][1]	XXXX XXXX XXXX XXXX XXXX XXXX XXX0 0010$_B$
adc_results[1][1]	XXXX XXXX XXXX XXXX XXXX XXXX XXX0 0110$_B$
adc_results[2][1]	XXXX XXXX XXXX XXXX XXXX XXXX XXX0 1010$_B$
adc_results[3][1]	XXXX XXXX XXXX XXXX XXXX XXXX XXX0 1110$_B$
adc_results[4][1]	XXXX XXXX XXXX XXXX XXXX XXXX XXX1 0010$_B$
adc_results[5][1]	XXXX XXXX XXXX XXXX XXXX XXXX XXX1 0110$_B$
adc_results[6][1]	XXXX XXXX XXXX XXXX XXXX XXXX XXX1 1010$_B$
adc_results[7][1]	XXXX XXXX XXXX XXXX XXXX XXXX XXX1 1110$_B$

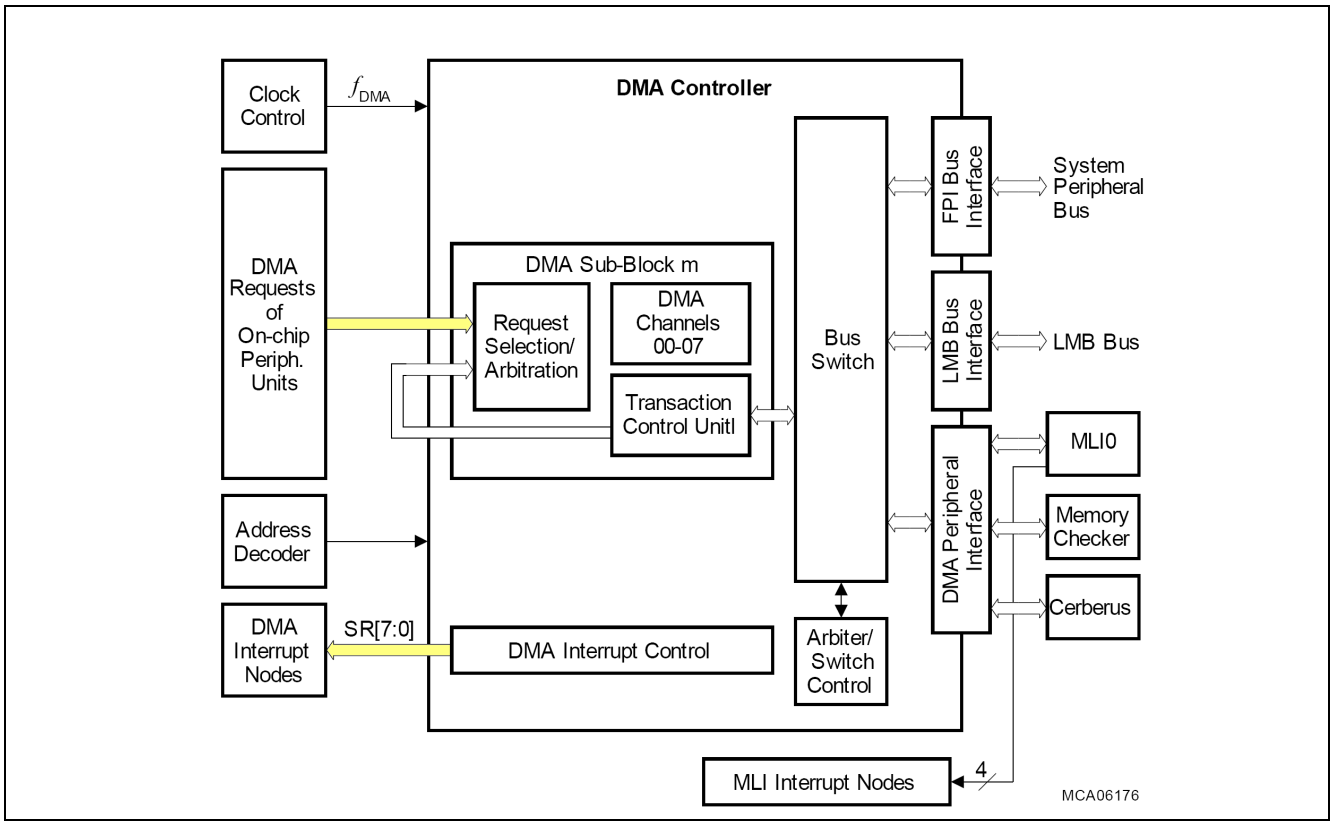

Figure 7 DMA Module Implementation and Interconnections. Request line from ADC and to TriCore core maked yellow.

3.4 Emergency Stop Output Control

The emergency stop feature of the TC1782 allows for a fast emergency reaction on an external event without the intervention of software. In an emergency case, the PWM signal outputs can be selectively put immediately to a well-defined logic state. The emergency case is indicated by an emergency input signal with selectable polarity that has to be connected to input P1.4.

Figure 8 shows a diagram of the emergency stop input logic. This logic is controlled by the SCU Emergency Stop Register EMSR.

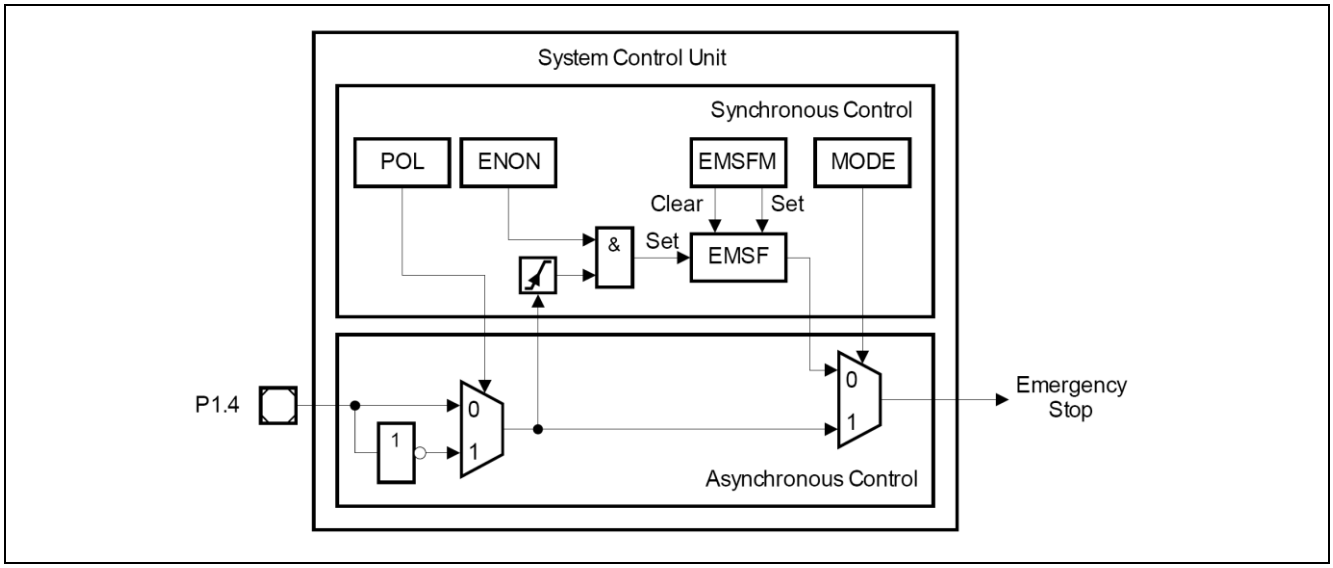

Figure 8 Emergency Stop Input Control

In the example application with DEBUG defined P1.5 is set to high and connected to P1.4 with a jumper on the TriBoard. Listing 4 shows the complete EMGSTOP initialization starting with the emergency stop level for the PWM signal at port 0 and 1 (Line 338-339). The emergency stop input port pin P1.4 is configured with an internal pull down device. Emergency stop is enabled for the PWM output pins (Line 350-351). The last line (Line 352) enables the module with synchronized control.

```
335   /*
336    * Emergency Stop (EMGSTOP)
337    */
338   P0_OUT.U = 0; // P0.8, P0.9 emergency stop to low level
339   P1_OUT.U = 0; // P1.2, P1.3, P1.10, P1.11 emergency stop to low level
340 #ifdef DEBUG
341
342   P1_IOCR4.U = 0x00008010; // P1.5 OUT, P1.4 input pull-down device for the
    emergency stop input signal
343   P1_OMR.B.PS5 = 1; // Set P1.5
344 #else
345
346   P1_IOCR4.U = 0x00000010; // P1.4 input pull-down device for the emergency
    stop input signal
347 #endif
348
349   endinit_clear(WDT_DISABLED);
350   P0_ESR.U = 0x300; // Emergency stop enable for P0.8 and P0.9
351   P1_ESR.U = 0xC0C; // Emergency stop enable for P1.2,P1.3,P1.10 and P1.11
352   SCU_EMSR.U = 0x02000005; // Falling edge enables EMSF. The clocked path
    directly from the input pin is selected, clear EMSF
353   endinit_set(WDT_DISABLED, WDT_DISABLED);
```

Listing 4 Emergency Stop Initialization

4 Example Application

The example application configures the hardware after reset as described in chapter 3 and enters an endless loop. The motor rotation is simulated by incrementing a global angle variable. In a real application this is angle value is updated by the position sensor interface. The TriCore interrupt routine implements a basic motor control algorithm (Listing 5) consisting of the load operation of the phase currents from the DMI RAM, an inverse park transformation from the angle modified in the main loop, a space vector modulation and an update of the GPTA LTC registers for the PWM update.

The control algorithm uses an optimized implementation of the Park- and Clarke transformation and a space vector modulation (Listing 6) described in [5]. The functions are available as C code and in assembler. The output of the space vector modulation is shown in Figure 9. The signal output is shown in Figure 11.

The total execution time for the control algorithm is about 150 CPU cycles 1.7% CPU load.

```
100 void __interrupt (PWM_INT) pwm_isr(void)
101 {
102   SVGENDQ s;
103   IPARK ip = { 0.0, (__fract) FRACT_MAX };
104   PARK p;
105   CLARKE c;
106   RESOLVER r;

121   c.As = adc_results[2][0]; // Iu
122   c.Bs = adc_results[2][1]; // Iw
123   clarke_calc(&c);
124   r.SinIn = adc_results[1][0]; // Res_A
125   r.CosIn = adc_results[1][1]; // Res_B
126   resolver_calc(&r);
127   p.Angle = rotor_theta;
128   p.Alpha = c.Alpha;
129   p.Alpha = c.Beta;
130   park_calc(&p);
131   // TODO do more
132   ip.Alpha = p.Alpha;
133   ip.Alpha = p.Beta;
134   ip.Angle = p.Angle;
135   ipark_calc(&ip);
136   s.Ualpha = ip.Alpha;
137   s.Ubeta = ip.Beta;
138   svgendq_calc(&s);
139   pwm_update(&GPTA0_LTCCTR04.U, &GPTA0_LTCXR08.U, s.Ta, s.Tb, s.Tc);

156   DMA_INTCR.U = 0x00000002; // clear the interrupt event(s)
157 }
```

Listing 5 Control algorithm and PWM update

```
#define SQT3   1.7320508075 // SQT(3)
struct  {
  __fract svgendq_coeffs[24];
  char svgendq_lookup[6];
  __fract qseed3;
} __near dmc = {{
  +.5,-SQT3/2, .5, SQT3/2,+1 ,    0,-.5,-SQT3/2,
  -1 ,    0, .5,-SQT3/2,-.5,-SQT3/2,-.5, SQT3/2,
  +.5, SQT3/2,-1,     0,-.5, SQT3/2,+1 ,    0},
  {2*0, 2*4, 2*5, 2*2, 2*1, 2*3},
  1/SQT3};

typedef struct {
  __fract Ta,Tb,Tc;    // Output: phase-a, b, c switching function
  __fract Ualpha, Ubeta; // Input:  alpha, beta-axis phase voltage
} SVGENDQ;

typedef SVGENDQ *SVGENDQ_handle;

inline void svgendq_calc(SVGENDQ_handle v) {

  int s;
  __fract __circ *circ_ptr;
  __fract t1, t2, t, tt, *p;
  __fract Ubeta = v->Ubeta;

  Ubeta *= dmc.qseed3;
  s = Ubeta >= 0;
  s = Ubeta < v->Ualpha ? s + 2 : s;
  Ubeta = -Ubeta;
  s = v->Ualpha < Ubeta ? s + 4 : s;
  s = s ? s : 1;
  s = dmc.svgendq_lookup[s-1];

  circ_ptr = __initcirc(v,3*sizeof(__fract), s & 0xC);

  p = &dmc.svgendq_coeffs[s*2];
  t1 = p[0] * v->Ubeta + p[1] * v->Ualpha;
  t2 = p[2] * v->Ubeta + p[3] * v->Ualpha + t1;
  t = (dmc.svgendq_coeffs[4]-t2)>>1; // t =(1-t2)/2

  s &= 2;
  tt = !s ? t + t1 : t;              // force conditional arithmetic
  tt = s ? tt + t2 : tt;
  *circ_ptr++ = tt;
  tt = s ? t + t1 : t;
  *circ_ptr++ = t;
  t = !s ? tt + t2 : tt;
  *circ_ptr = t;
}
```

Listing 6 Space Vector Modulation – C Source Code

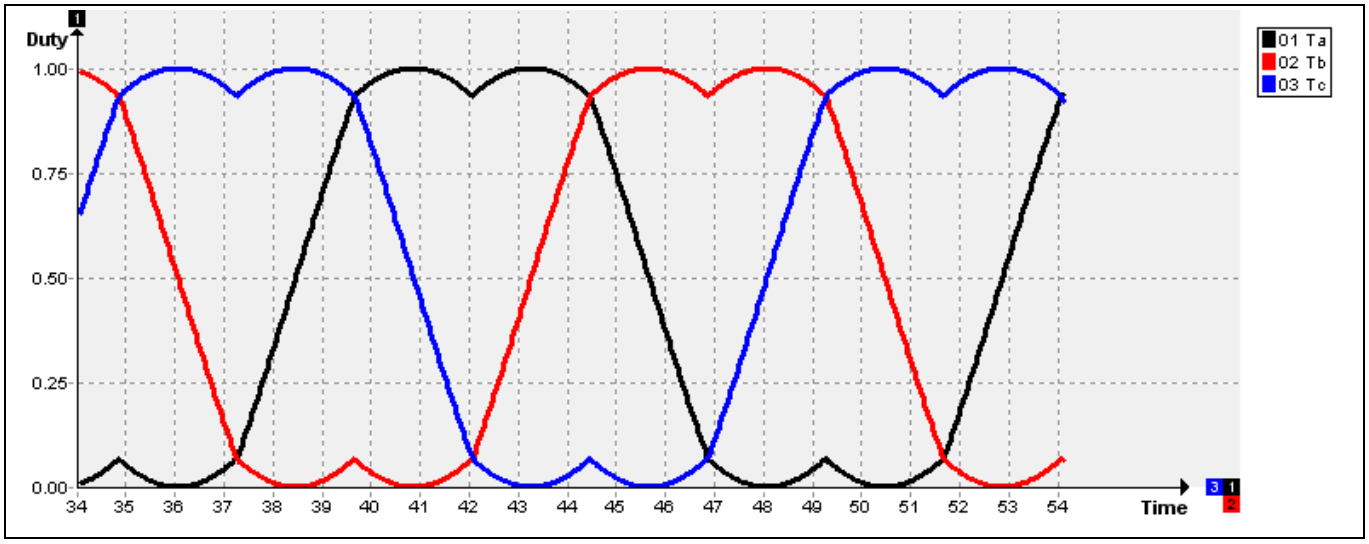

Figure 9 Space Vector Modulation

```
adc_results = (
  (0.0, 0.032227),
  (0.034363, 0.034485),
  (0.034454, 0.034546),
  (0.069031, 0.03476),
  (0.034302, 0.034363),
  (0.034241, 0.034302),
  (0.034332, 0.034332),
  (0.034271, 0.034302),
  (0.03418, 0.03421))
```

Figure 10 Debugger Watch output of ADC result array

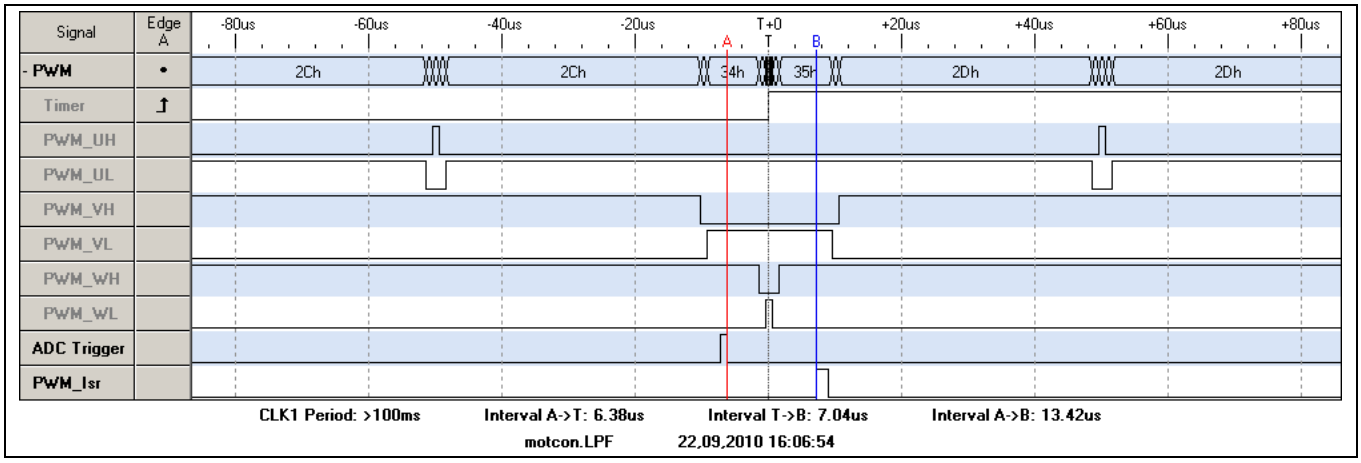

Figure 11 Logic Analyzer showing 3-phase PWM

Figure 11 display the PWM signal output on a logic analyzer with a scan from channel 9 to 0. Seven ADC conversions of 0.91 µs measured from the falling edge on ADC_TRIGGER to the PWM start is 6.38 µs (Interval A->T). Three additional DC conversions and the two DMA transfers required 7.04 µs (Interval T->B). I.e the 2 DMA transfers require about 4.31 µs. The start and end of the control algorithm is given by the PWM_Isr signal and measured to 1.96 µs.

5 Tools

The examples were build using the Tasking compiler Version 3.5r1. mingw32-make (www.mingw.org) was used as a make tool. The example code includes a project workspaces for the PLS UDE debugger V3.0.8.

6 Source code

The source code provided with this application consists of a single Tasking project.

7 References

[1] http://www.infineon.com/tricore

[2] TriCore Architecture V1.3.8 2007-11

[3] TC1784 User's Manual V1.0 2009-07

[4] Application Note AP32084, TriCore Sinusoidal 3-Phase Output Generation Using the GPTA

[5] Application Note AP32135, TriCore 3-phase complementary PWM with hardware triggered ADC conversion

[6] Application Note AP16084 , Field Oriented Control of a PMSM using a Single DC Link Shunt

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

Published by Infineon Technologies AG