NXP Semiconductors
Application Notes

Document Number: AN5374
Rev. 0, 12/2016

eTPU library usage in an application

Covers all Automotive MCUs with eTPU, eTPU2 and eTPU2+

by: Marketa Venclikova

Contents

1. Introduction

The Enhanced Time Processor Unit (eTPU) is an autonomous, CPU independent 24-bit programmable I/O controller having its own core and memory system allowing to perform complex timing and I/O management.

The eTPU was originally designed for combustion engine and advanced timing control, but nowadays alongside with expansion of hybrid vehicles eTPU is gaining momentum in the field of automotive motor control applications as well.

A new eTPU/eTPU2/eTPU2+ library of various functions for engine control and motor control has been developed and is provided in the form of precompiled binaries (eTPU function set) available to download from http://www.nxp.com/webapp/etpu_cw/. The new library is based on the previous eTPU function sets (set1 to set4), where most of the functions were updated, added new functionality or completely new functions were implemented.

This application note serves as a cookbook on how to get eTPU binaries and related files and embed them into an application. For better understanding an example how to generate PWM signal using one of the eTPU Motor Control library functions is provided in section 4.

2. eTPU library overview

NXP provides an eTPU library comprising functions for engine control, motor control and general timing based on CodeWarrior for eTPU development tool. Following functions are already available to download:

- Engine control functions
  - Cam
  - Crank
  - Crank Emulator
  - Tooth Generator
  - Fuel
  - Direct Injection
  - Spark
  - Knock Window
- Motor Control:
  - Motor Control PWM
  - Resolver Interface

All of the above listed functions are compatible with eTPU, eTPU2 and eTPU2+. Other functions intended for general timing and communication are planned to be included in the eTPU library. This application note only provides instruction how to incorporate the eTPU Library functions binaries into an user application. A detailed description of individual functions can be found in AN4907 (Engine Control eTPU Library) and in AN5335 (Motor Control eTPU Library). Table 1 presents several ways how to get the eTPU Function code, depending on whether you want a ready to use eTPU functions with precompiled binaries (follow the rows for an eTPU user), or there is a need for you to modify the eTPU function source code (follow the rows for an eTPU developer).

Table 1 Where to get the eTPU function code and the tools needed

	How to get the code	Tool needed
eTPU user	CW eTPU Function Selector – custom eTPU binary image	-
	AN4908 Engine Control demo SW	GHS Multi compiler
	AN5353 Motor Control eTPU Demo S	S32DS
eTPU developer	AN4907 Engine Control eTPU Library	Code Warrior for eTPU
	AN5335 Motor Control eTPU Library	
	eTPU Library ported to ASH WARE	ASH WARE development tool

3. How to get eTPU function binaries

This chapter describes how to download eTPU function binary and how to include it into a project and get the application running using eTPU functions.

3.1. Download the eTPU function set

User can easily download the required eTPU function using a designated eTPU function selector web tool available at http://www.nxp.com/etpu. The first step is to select the corresponding version of eTPU that the application will be running on and select particular function/functions to be downloaded. Functions marked as “Included routines” are automatically included to eTPU function set according to selection made. Devices supporting particular version of the eTPU are highlighted (see Figure 1 eTPU function selector web tool), eTPU2 and eTPU2+ versions are back compatible with functions intended for eTPU, but not conversely.

As a second step the user is prompted to provide a feedback describing the features of the application where eTPU functions are to be used in and give a short overview about the eTPU tasks.

The eTPU function set will be compiled in the third step by clicking on the “Create function set” button and user will be able to download it as a zip file.

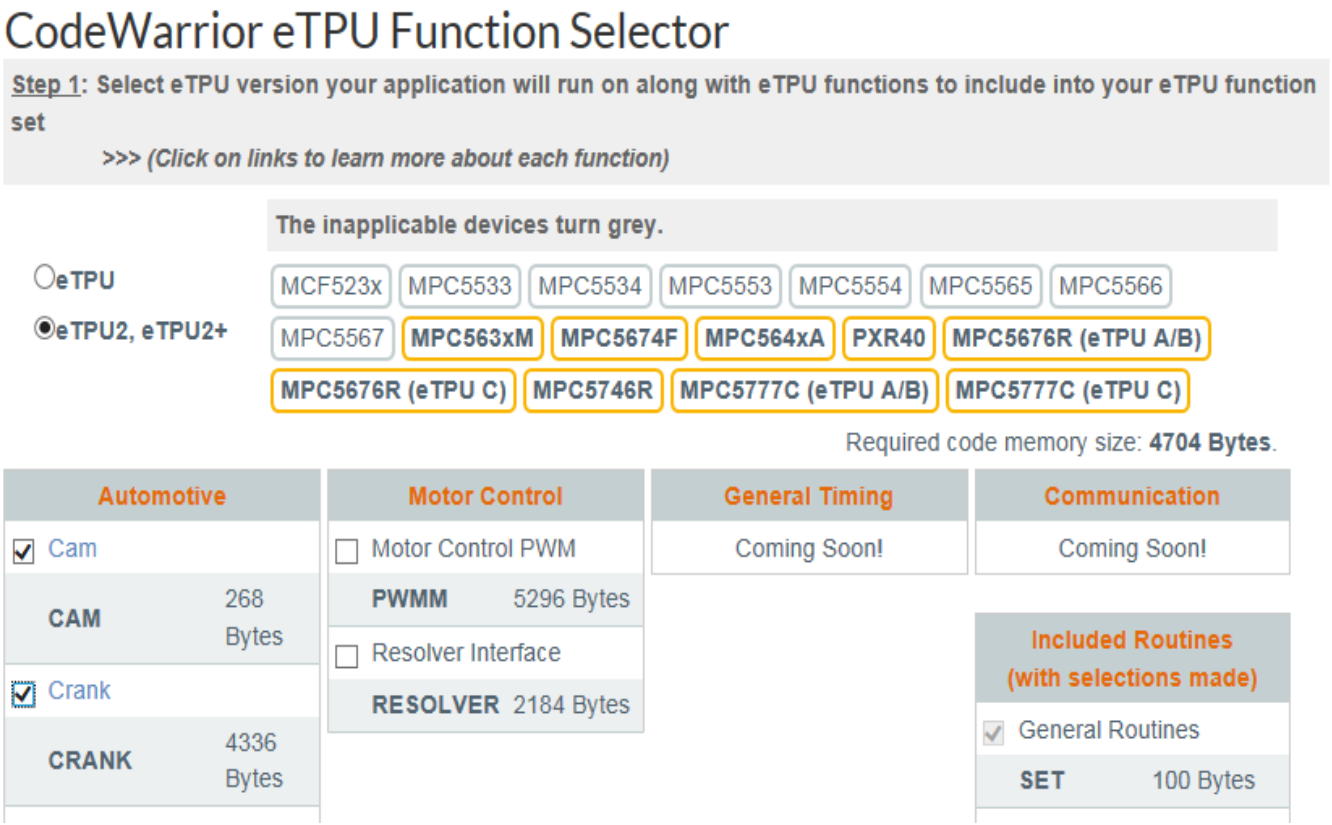

Figure 1 eTPU function selector web tool

3.2. The etpu_set content overview

When unpacked the file “etpu_set.zip” contains following folders and files:

- ***etpu*** – this folder includes other subfolders with C utility functions, API and binary image of the eTPU function.
  - The ***_utils*** folder contains files *etpu_util.c /.h* including low-level functions handling the eTPU.
  - The ***etpu_set*** folder contains the binary image of eTPU code and interface files, all generated by the eTPU compiler.
  - The **<*func*>** folder(s), <*func*> represents the function(s) tag, include the eTPU function’s API. There is one separate folder for each eTPU function. These drivers enable an easy access to eTPU functions from an application.
- ***include*** – this folder contains device specific header files, one of which needs to be included in the user *main.c* file and *etpu_struct.h* file defining the eTPU registers – an essential file for etpu_util.h /.c.
- ***files*** – *etpu_gct.c_* and *etpu_gct.h_* are eTPU configuration templates.

The user is strictly recommended not to modify any file from *etpu* and *include* folders, functions are ready to be used in a user application by a function call. The only files allowed to be modified are *etpu_gct.c* and *etpu_gct.h*.

Table 2 content of the *etpu_set* folder overview

Folder	Subfolder	Content
etpu	*_utils*	*etpu_util.c* *etpu_util.h*
	etpu_set	*etpu_<func>_auto.h* *etpu_set.h*
	<func>	*etpu_<func>.c* *etpu_<func>.h*
include	-	*<device>_vars.h* *typedefs.h* *etpu_struct.h*
-	-	*etpu_gct.c_* *etpu_gct.h_*

3.3. The eTPU project structure

There is a structure of the project using eTPU functions on Figure 2 that have to be followed by user when including eTPU function binaries, eTPU APIs and eTPU Utilities into an application. All these items are included in the etpu_set.zip download package.

The eTPU source code (at the bottom of the Figure 2) is not included in the etpu_set.zip package. The source code is available with AN4907 (Engine Control eTPU Library) and AN5335 (Motor Control eTPU Library). Modification of the source code requires eTPU development tools (e.g. CodeWarrior, AshWare) and specific knowledge of eTPU programming.

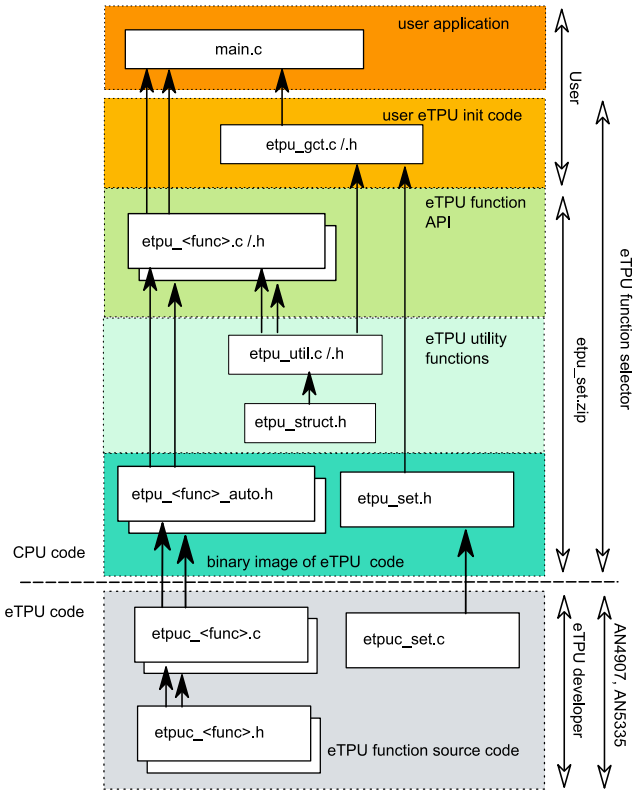

Figure 2 Structure of a project utilizing eTPU functions

3.4. The eTPU initialization

To correctly initialize the eTPU user is recommended to use *etpu_gct.c /.h* template files. The templates include eTPU initialization code ready to be modified specifically for the user application. The user needs to include *etpu_gct.c /.h* and *etpu_<func>.c /.h* into the *main.c* of the application, add function calls *my_system_etpu_init/start* to configure the eTPU and create an eTPU interrupt service routine including eTPU run-time interface code. Perform the following steps:

1. Include *etpu_gct.c /.h* template files into *main.c* and modify them
2. Include *etpu_<func>.c /.h* API into the *main.c*.

3. Create an eTPU interrupt routine utilizing the eTPU function run-time interface code if needed
4. Call the *my_system_etpu_init();* and *my_system_etpu_start();* in the main function to initialize eTPU

User can optionally include FreeMASTER binaries for debug purposes or standard automotive Motor Control Library (MCLib) if some math operations needed.

4. The eTPU PWMM function utilization in an application

The MPC574xR controller board is used for the demonstration of the eTPU PWMM motor control function. The demo application is created in S32 Design Studio (S32DS), available to download free at http://www.nxp.com/s32ds. The S32DS is supported for MPC564xA, MPC5676R, MPC574xR and MPC5777C devices having eTPU2 and eTPU2+. For the debug and demonstration purpose the FreeMASTER Run-Time Debugging Tool (available to download free at http://www.nxp.com/freemaster) is used to visualize PWMM outputs. Finally, several functions from Automotive Math and Motor Control Library Set are used within this application.

4.1. PWMM function overview

This eTPU Motor Control PWM function provides various options of PWM configuration. Depending on configuration it utilizes either four or seven eTPU channels - three channels to generate three phases of single output PWM signal, or six eTPU channels to generate three phases of complementary output PWM signals; plus one channel is always used to update and synchronize all the eTPU PWM outputs.

The function API utilizes four structure types, where function configuration, data, states and inputs are stored. All of the structures need to be defined within the configuration file *etpu_gct.c*. The structure types are as follows:

- pwmm_instance_t – this structure holds parameters used to initialize the eTPU functions, such as numbers of channels assigned to phases and master channel, type of phases (single/complementary), PWM polarity etc.
- pwmm_config_t – this structure holds configuration parameters of the modulation, that are allowed to be changed in run-time.
- pwmm_inputs – this structure holds values which are inputs driving the PWM duty-cycles
- pwmm_states – this structure holds states and actual duty cycles of the PWM output signals

The PWMM API includes the following routines to handle the PWMM function running on the eTPU channels:

- fs_etpu_pwmm_init()
- fs_etpu_pwmm_config()
- fs_etpu_pwmm_set_inputs()
- fs_etpu_pwmm_enable()
- fs_etpu_pwmm_disable()
- fs_etpu_pwmm_get_states()

For more detailed description please refer to PWMM-doxyDoc file included together with the eTPU PWMM function API source code.

4.2. Download the PWMM function

To get the eTPU PWM motor control function, the eTPU binaries and adjacent files need to be downloaded, as described in chapter 3.1. Together with the PWMM function the binary image of the MC_SIN and MC_UTIL are included. The generated etpu_set.zip file contains following subfolders:

- etpu
  - _etpu_set
  - _utils
  - pwmm
- include
- templates *etpu_gct.c_ /.h_*

4.3. Create a S32DS project

As soon as S32 Design Studio is properly installed (version 1.1 for Power Architecture used in this example), a new project can be created performing following steps:

1. Select **File → New → S32DS Project**
2. Type a name of the project to the upper line of the project wizard window and select the appropriate device on the left panel, in this application particularly MPC5743R and continue with **Next.** The device selection can be seen on Figure 3 Project wizard - selection of the device.
3. Then select language (C/C++) and choose the debugger option.
4. Click **Finish** to complete the project creation. The new project will appear on the left bar in the project explorer.

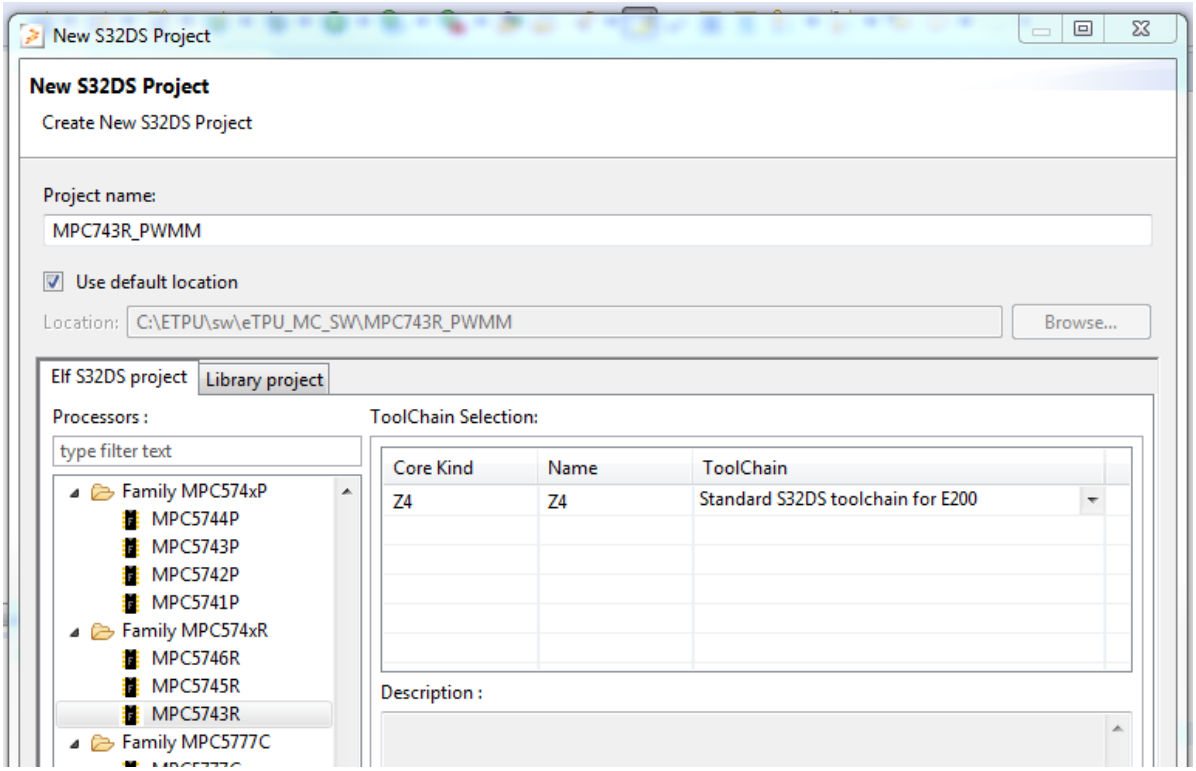

Figure 3 Project wizard - selection of the device

4.4. Include files into a project

All the downloaded eTPU PWMM function related files need to be included into the new S32DS project. Do following steps:

- Unzip the *etpu_set.zip* and copy paste the content as is to the “src” in the project.
- Delete all the redundant header files from “etpu_set\include” except of *etpu_struct.h* and *mpc5746r_vars.h* . Other header files are redundant.
- Retype the ending of etpu_gct.c_ and etpu_gct.h_ to .c and .h in “etpu_set”
- FreeMASTER and Automotive Math and Motor Control library (MCLIB) are utilized within this project – source codes and communication drivers are pasted in the “src” folder in the project.

Finally include following header files in the main_Z4_1.c file:

```
/*****************************************************************************
* User includes
*****************************************************************************/
#include "etpu_set\etpu\pwmm\etpu_pwmm.h"  /* eTPU PWMM function API */
#include "etpu_set\etpu_gct.h"   /* eTPU configuration */
#include "MCLIB\gmclib.h" /* General Motor Control Library */
#include "MCLIB\gdflib.h" /* General Motor Control Library */
#include "FreeMASTER\MPC57xx\freemaster.h" /* FreeMASTER Communication driver */
```

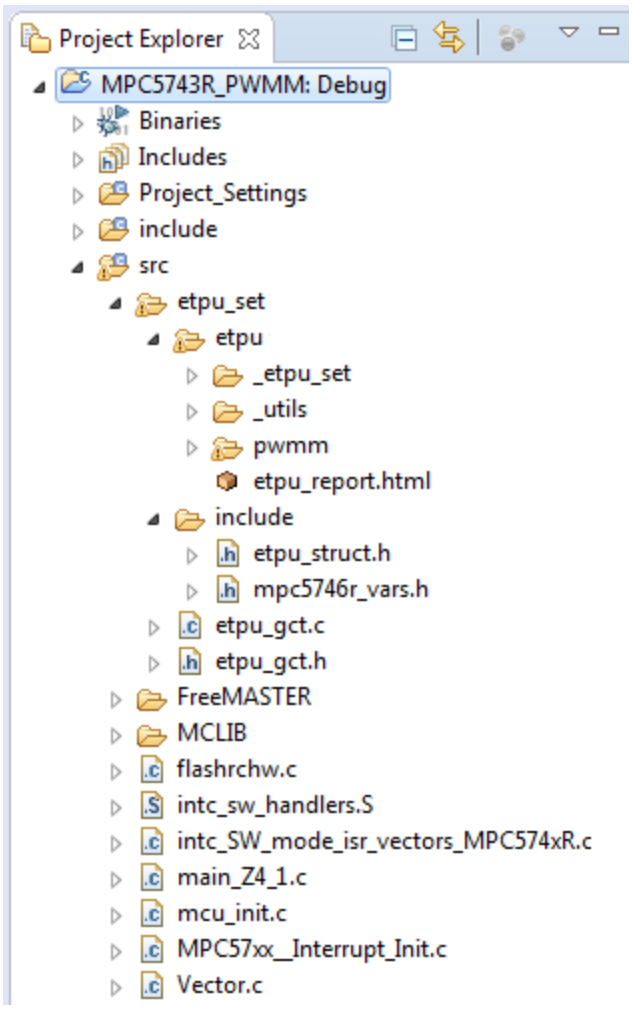

Figure 4 Project final content

4.5. Configuration of the ETPU

The eTPU configuration can be easily done modifying *etpu_gct.c* and *etpu_gct.h* files. Thanks to those API files the user is prevented from direct access to the eTPU registers.

4.5.1. Edit *etpu_gct.h*

The eTPU channel functionality definition in this application is done in *etpu_gct.h* as follows:

- **Assign eTPU functions to eTPU channels** – master channel does not generate any output signal, this channel is intended for eTPU PWMM output synchronization. In this particular application complementary PWM signals are configured, so it means that one phase is driven by two eTPU channels. Note that the channel numbers assigned to one phase must be consecutive.

```
/*****************************************************************************
* Define Functions to Channels
*****************************************************************************/
#define ETPU_PWMM_MASTER_CH             ETPU_ENGINE_A_CHANNEL(10)
#define ETPU_PWMM_PHASEA_BASE_CH        ETPU_ENGINE_A_CHANNEL(11)
#define ETPU_PWMM_PHASEA_COMPL_CH       ETPU_ENGINE_A_CHANNEL(12)
#define ETPU_PWMM_PHASEB_BASE_CH        ETPU_ENGINE_A_CHANNEL(17)
#define ETPU_PWMM_PHASEB_COMPL_CH       ETPU_ENGINE_A_CHANNEL(18)
#define ETPU_PWMM_PHASEC_BASE_CH        ETPU_ENGINE_A_CHANNEL(15)
#define ETPU_PWMM_PHASEC_COMPL_CH       ETPU_ENGINE_A_CHANNEL(16)
```

- **Enable interrupt from eTPU channels** – interrupt flag is set by the Master Channel every time when there is a missing update on any of the three phases. In order to enable the interrupt flag propagation to the interrupt controller (INTC), a bit corresponding to the PWMM master channel number must be set in the CIE register.

```
/*****************************************************************************
* Define Interrupt Enable, DMA Enable and Output Disable
*****************************************************************************/
#define ETPU_CIE_A (1<<ETPU_PWMM_MASTER_CH) /* enable interrupt on ETPU PWMM_CHAN */
```

- **Enable global variable access** – make the eTPU structures visible outside the *gct* to be accessible also for PWMM API.

```
/*****************************************************************************
* Global Variables Access
*****************************************************************************/
/* Global <FUNC1> structures defined in etpu_gct.c */
extern struct pwmm_instance_t pwmm_instance;
extern struct pwmm_config_t  pwmm_config;
extern struct pwmm_inputs_t pwmm_inputs;
extern struct pwmm_states_t pwmm_states;
```

- **Define the eTPU system frequency –** update the value of the macro to be corresponding to the eTPU system frequency. All of the eTPU timing is derived from the system clock so it is necessary to specify it here.

```
/*****************************************************************************
* Application Constants and Macros
*****************************************************************************/
#define SYS_FREQ_HZ  160E6
```

4.5.2. Configure *etpu_gct.c*

Configuration of the eTPU together with eTPU function structures definition (in particular PWMM related structures) is done in *etpu_gct.c*. The eTPU itself configuration structures are already part of this file and serves as a template of eTPU configuration. Function specific structures need to be defined manually here in this configuration file.

- **Include device specific header file to specify eTPU data RAM frame**

```
#include "mpc5746r_vars.h"
```

- **Configure the eTPU engine -** modify the predefined structure including parameters to configure eTPU engine. For a detailed description of the eTPU engine configuration please refer to eTPU Reference Manual.

```
struct etpu_config_t my_etpu_config =
	{
		…
	};
```

- **Define PWMM instance –** assign the particular channels numbers using the macros defined in etpu_gct.h, set the priority of the eTPU function, specify the type of the phases (single/complementary), PWM modulation, polarity, function start offset and pointer to a channel parameter base address. All the four eTPU PWMM structure types are declared and described in the *etpu_pwmm.h* API.

```
struct pwmm_instance_t pwmm_instance =
	{
		ETPU_PWMM_MASTER_CH,	/**< Channel number of the PWMM master channel. */
		ETPU_PWMM_PHASEA_BASE_CH, /**< Channel number of the Phase A channel. */
		ETPU_PWMM_PHASEB_BASE_CH, /**< Channel number of the Phase B channel. */
		ETPU_PWMM_PHASEC_BASE_CH, /**< Channel number of the Phase C channel. */
		FS_ETPU_PRIORITY_MIDDLE,  /**< Channel priority for all PWMM channels. */
		FS_ETPU_PWMM_FM0_COMPLEMENTARY_PAIRS,	/**< Type of phases. */
		FS_ETPU_PWMM_POLARITY_BASE_ACTIVE_HIGH
		|FS_ETPU_PWMM_POLARITY_COMPL_ACTIVE_HIGH,	/**< Base and complementary
		channel polarity */
		(uint24_t)USEC2TCR1(100),
		0			/**< Channel parameter base address */
	};
```

- **Define PWMM configuration structure** – specify the type of PWMM modulation, the modulation mode (center aligned/left aligned/…), period time, dead time, minimum pulse width and update time. All the timing parameters are expressed in TCR1 cycles. The macros from etpu_gct.h can be used to transfer mili/micro/nano-seconds to a number of TCR1 cycles. Parameters of this structure can be changed during run-time.

```
struct pwmm_config_t pwmm_config =
	{
		FS_ETPU_PWMM_FM1_FRAME_UPDATE_ONLY,	/**< selection of PWM update position.*/
		FS_ETPU_PWMM_MODULATION_SINE_TABLE,	/**< Selection of modulation. angle =
		input_a, amplitude = input_b*/
		FS_ETPU_PWMM_MODE_CENTER_ALIGNED,	/**< PWM Mode selection */
		USEC2TCR1(50),	/**< PWM period as a number of TCR1 cycles. */
		USEC2TCR1(1),	/**< PWM deadtime as a number of TCR1 cycles.*/
		USEC2TCR1(1),	/**< Minimum pulse width as number of TCR1 cycles. */
		USEC2TCR1(20)	/**< A time period (number of TCR1 cycles) that is needed to
		perform an update of all PWM phases. */
	};
```

- **Define PWMM inputs structure –** this structure holds input parameters for phases A, B and C used for update of the generated duty cycles. This structure can remain empty during initialization.

```
struct pwmm_inputs_t pwmm_inputs;
```

- **Define PWMM states structure –** this structure contains outputs representing internal states of the PWMM function.

```
struct pwmm_states_t pwmm_states;
```

- **Implement the eTPU initialization functions –** the eTPU initialization is hold by 2 functions:
  - **my_system_etpu_init()**
    This function consists of 2 parts. The first one is eTPU engine configuration using my_etpu_init structure, the second one configures all eTPU channels using appropriate API calls. For PWMM, the call of fs_etpu_pwmm_init(…) needs to be added the channel configuration part.
  - **my_system_etpu_start()**
    This function does not need to be modified. It applies the interrupt-enable and DMA-enable masks and starts the eTPU internal clocks (TCR1 and TCR2).

```
/* Initialization of eTPU channel settings */
err_code = fs_etpu_pwmm_init(&pwmm_instance, &pwmm_config);
if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_PWMM_MASTER_CH<<16));
```

4.6. Get the eTPU running

Create a function that will perform update of the PWM phases by *fs_etpu_pwmm_config(&pwmm_instance, &pwmm_config, &pwmm_inputs);* function call from the PWMM API. Call this function every time after the last update was successfully performed (check the update status in the *pwmm_states* structure)

```
void etpu_pwmm_update(void)
{
	/* Local variable definition */
	/* … */

	/* User code */
	/* … */

	/* PWMM update */
	fs_etpu_pwmm_config(&pwmm_instance, &pwmm_config, &pwmm_inputs);
	fs_etpu_pwmm_get_states(&pwmm_instance, &pwmm_states);

	/* Check the eTPU load */
	etpu_engine_load = get_etpu_load_a();
}
```

Create an eTPU interrupt routine servicing the eTPU PWMM Master Channel interrupt. The interrupt flag on the Master Channel is set whenever there is a missing update on any of the PWM phases.

```
void etpu_pwmm_master_irq(void)
```

```
{
	/* Local variable definition */
	/* … */

	/* Clear the channel interrupt flag */
	fs_etpu_clear_chan_interrupt_flag(ETPU_PWMM_MASTER_CH);

	/* User code */
/* … */}
```

To properly initialize and start the eTPU, perform *my_system_etpu_init();* followed by *my_system_etpu_start();* function calls in the *main* function. Then start the PWM generation by *fs_etpu_pwmm_enable(&pwmm_instance)* API function call.

```
int main(void)
{
	/* Initialize the CPU */
	/* … */

	/* Initialize and run the eTPU */
	my_system_etpu_init();
	my_system_etpu_start();
	/* Enable the PWMM eTPU function */
	fs_etpu_pwmm_enable(&pwmm_instance);

	/* Loop forever */
	for(;;) {

	/* User code */
	/* … */
	}
}
```

4.7. Configure FreeMASTER and debug

For the debugging and PWM duty-cycle visualization purpose the FreeMASTER Run-Time Debugging Tool is used. Include the source code into your project and configure FreeMASTER in *freemaster_cfg.h* template header file. For proper settings please refer to FreeMASTER documentation included in the FreeMASTER installation package.

4.8. Application output

In this example the Sine Table PWM modulation utilizing complementary channel pairs is configured, the duty cycle time change of the three phases can be seen on Figure 5. Each curve represents the value of duty cycle change during time on one particular phase.

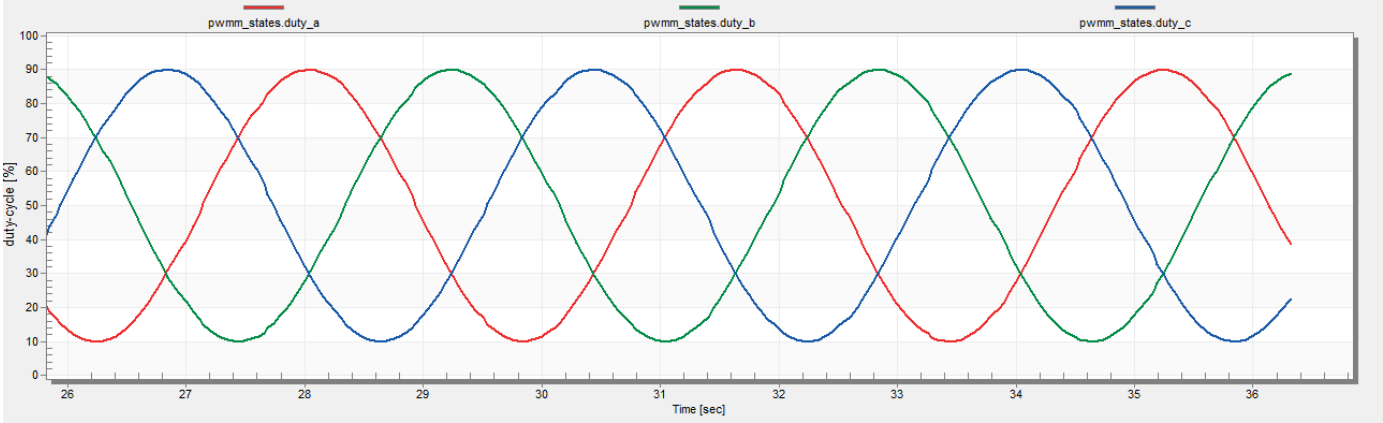

Figure 5 Sine table PWM modulation duty cycles

In Figure 6 the output PWM signals are displayed using oscilloscope. There are together 6 PWM outputs, two outputs per phase (base and complementary channel).

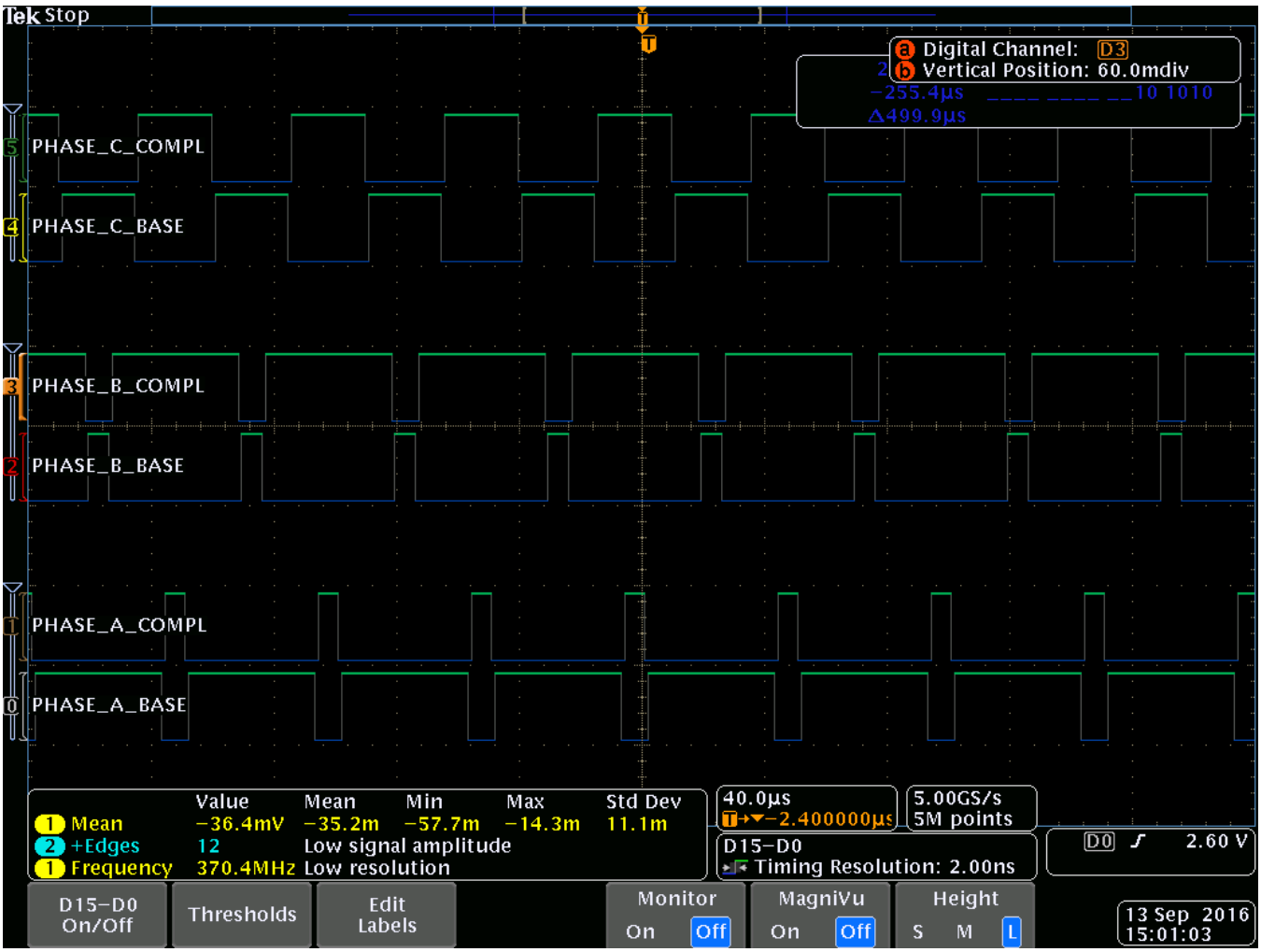

Figure 6 PWM outputs displayed on osciloscope

5. Summary

In this application note instructions on how to get the eTPU up and running using the available NXP eTPU libraries are given. A detailed description of the software package generated by the eTPU Function Selector and its integration into a user application is provided. Moreover, a particular example application running the eTPU PWMM function on MPC5746R is described and included.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN5374
Rev. 0
12/2016