Freescale Semiconductor
Application Note

Document Number: AN4908
Rev. 1, 08/2015

Engine Control eTPU Demo Application

Runs on MPC5674F evaluation board and demonstrates the usage of Engine Control eTPU Library

by: Milan Brejl

1. Introduction

The Enhanced Time Processing Unit (eTPU) is a programmable I/O controller with its own core and memory system, allowing complex timing and I/O management independent of the CPU. The eTPU module is a peripheral for 32-bit Qorivva powertrain families of MPC5500, MPC5600, and MPC5700.

The eTPU is used as a co-processor, specialized for advanced timing functions and handles complex engine control, motor control, or communication tasks independent of the CPU. The engine management is the most traditional area of eTPU usage.

Contents

A new complex library of eTPU functions enabling the eTPU to drive engine control applications was developed in 2014 as described in [1].

This application note demonstrates a typical usage of all the Engine Control eTPU Library functions, API calls, initialization, run-time control, and monitoring. This application is dedicated for MPC5674F and features a FreeMASTER graphical interface on a PC.

2. Hardware

The hardware used to run the described demo application is listed in the table below.

Table 1 Hardware

Item	ID
Mother board	Evaluation Base Board MPC567XEVBMB
Daughter card	Demonstration Module MPC567XADAT516
MCU	MPC5674F
PC	A Windows-based PC connected via RS232 to the mother board, running FreeMASTER

The figure shows the evaluation board and required connections.

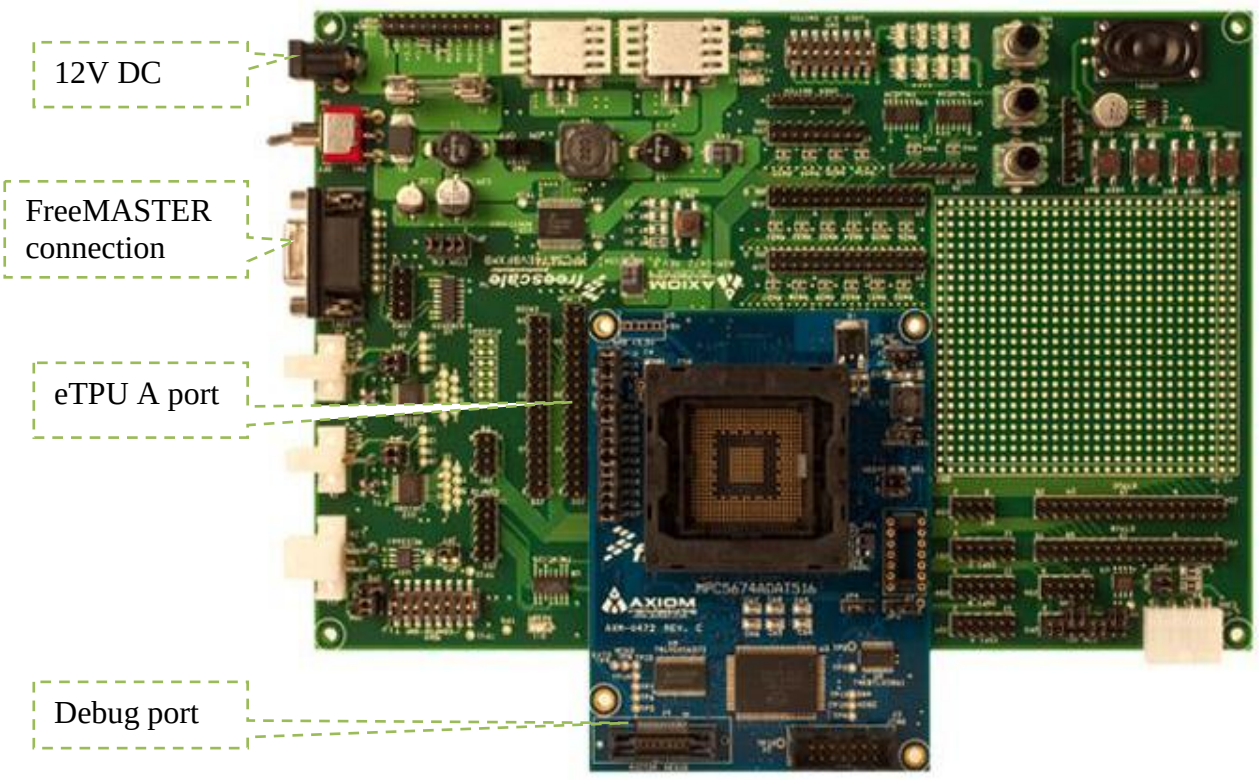


Figure 1 MPC5674F evaluation board and application connections

The assignment of Engine Control eTPU Library functions to eTPU channels and corresponding signals are listed in the table below.

Table 2 Assignment of eTPU functions to channels

Signal	eTPU A channel	Direction	ETPUA port (J22) pin
Cam	0	Input	1
Tooth Generator - Cam	1	Output	2
Crank	2	Input	3
Tooth Generator - Crank	3	Output	4
Spark – cylinder 1	4	Output	5
Spark – cylinder 2	5	Output	6
Spark – cylinder 3	6	Output	7
Spark – cylinder 4	7	Output	8
Fuel – cylinder 1	8	Output	9
Fuel – cylinder 2	9	Output	10
Fuel – cylinder 3	10	Output	11
Fuel – cylinder 4	11	Output	12
Knock – cylinders 1 and 4	12	Output	13
Knock – cylinders 2 and 3	13	Output	14
Direct Injection – bank 1	14	Output	15
Direct Injection – bank 2	15	Output	16
Direct Injection – cylinder 1	16	Output	17
Direct Injection – cylinder 2	17	Output	18
Direct Injection – cylinder 3	18	Output	19
Direct Injection – cylinder 4	19	Output	20

The hardware connection necessary to run the demo application can be easily done using two jumpers on the mother board eTPU A port (J22) listed in the table below.

Table 3 Evaluation board connections

Source	Destination
J22:2	J22:1
J22:4	J22:3

This connects the Tooth Generator outputs to the Cam and Crank inputs.

All application signals are available on the eTPU A port (J22). The complete assignment is listed in the table below.

Table 4 eTPU A port (J22) assignment

Cam	1	2	Tooth Generator - Cam
Crank	3	4	Tooth Generator - Crank
Spark - Cylinder 1	5	6	Spark - Cylinder 2
Spark - Cylinder 3	7	8	Spark - Cylinder 4
Fuel - Cylinder 1	9	10	Fuel - Cylinder 2
Fuel - Cylinder 3	11	12	Fuel - Cylinder 4
Knock – cylinders 1 and 4	13	14	Knock – Cylinders 2 and 3
Direct Injection – Bank 1	15	16	Direct Injection – Bank 2
Direct Injection – Cylinder 1	17	18	Direct Injection – Cylinder 2
Direct Injection – Cylinder 3	19	20	Direct Injection – Cylinder 4
	21	22	
	23	24	
	25	26	Test Pad – Tooth Generator
Test Pad – Crank	27	28	Test Pad - Cam
Test Pad - Spark	29	30	Test Pad - Fuel
Test Pad - Knock	31	32	Test Pad – Direct injection
	33	34	
+3.3 V	35	36	GND

The Test Pads can be used to monitor the timing of interrupt activity, as described in section eTPU run-time interface.

3. Software

The demo application is implemented using CodeWarrior Development Studio 10.6. Starting from a new project for MPC5674F, several ready to use Freescale drivers are added to the ***Sources*** folder:

- **eTPU**

  This folder includes all eTPU related drivers:

  - The ***util*** folder includes the low-level functions for using eTPU, available in [4].
  - The ***cam***, ***crank***, ***fuel***, ***spark***, ***knock***, ***inj***, and ***tg*** folders include the eTPU functions' API. They are available in [1]. These drivers enable an easy access to eTPU functions from an application.
  - The ***etpu_set*** folder includes the binary image of eTPU code and interface files, all generated by the eTPU compiler. They are available in [1].

- **FreeMASTER**

  This folder includes the FreeMASTER serial communication driver, which is available in [6]. For the eSCI module on MPC5674F, use the MPC55xx platform-dependent files (*freemaster_MPC55xx.c*/*.h* and *freemaster.h*).

- **GPIO**

  This is a pad configuration driver released as [5].

As additional folders are added to the project structure, the compiler settings needs to be adjusted as follows:

Choose: *Project* > *Properties* > *C/C++ Build* > *Settings* > *PowerPC Compiler* > *Input*, and add the project's source folder "*${ProjDirPath}/Sources*" to *User Recursive Path*.

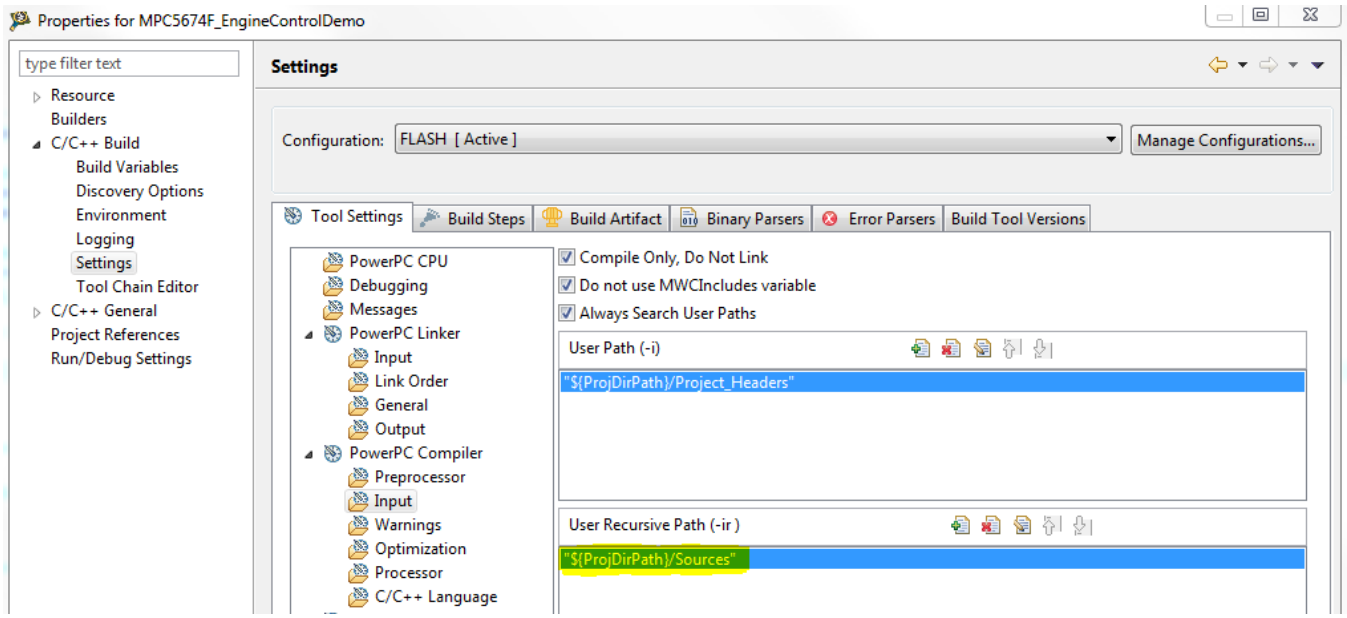


Figure 2 User Recursive Path

The source folder itself includes the code which demonstrates the usage of the Engine Control eTPU library. Among others, there are following more files:

- files *etpu_gct.c*/*.h*, which separates the eTPU initialization code, and
- files *main.c*/*.h*, which includes the top-level application code.

  Except the eTPU run-time interface, a very basic device initialization code is part of *main.c* file:

  - FMPLL is initialized to 100 MHz by the *fmpll_init()* function.
  - Selected GPIO pads are initialized as eTPU inputs and outputs by the *gpio_init()* function.

- eSCI A is configured to 115 kbd@100 MHz for FreeMASTER communication by the *esci_a_init()* function.
- Interrupt handlers are installed and interrupts are enabled by the *intc_init()* function.

The FreeMASTER operation is enabled by calling the following two routines from the FreeMASTER driver:

- *FMRST_Init()* at initialization time, and
- *FMSTR_Poll()* in the background loop.

The FreeMASTER operation is configured to run on background in polling mode without interrupting the application. The FreeMASTER Recorder feature is not supported.

3.1. eTPU initialization

The eTPU initialization is implemented in files *etpu_gct.c*/*.h* according to the template available in AN2864 (General C Functions for the eTPU).

The ***etpu_gct.h*** header file includes:

- Application constants with Crank pattern description and TCR2 angle base resolution:

```
/*****************************************************************************
* Application Constants and Macros
*****************************************************************************/
#define SYS_FREQ_HZ                                                    100E6
#define TCR1_FREQ_HZ                                          (SYS_FREQ_HZ/1)
#define TEETH_TILL_GAP                                                    35
#define TEETH_IN_GAP                                                       1
#define TEETH_PER_CYCLE                                                   72
#define TCR2_TICKS_PER_TOOTH                                            1000
#define TCR2_TICKS_PER_CYCLE          ((TEETH_PER_CYCLE)*(TCR2_TICKS_PER_TOOTH))
```

- Macros to simplify the conversion between raw values used by the eTPU and physical units:

```
#define MSEC2TCR1(x)                                    (TCR1_FREQ_HZ/1E3*(x)/1E0)
#define USEC2TCR1(x)                                    (TCR1_FREQ_HZ/1E3*(x)/1E3)
#define NSEC2TCR1(x)                                    (TCR1_FREQ_HZ/1E3*(x)/1E6)
#define DEG2TCR2(x)                                ((x)*TCR2_TICKS_PER_CYCLE/720)
#define UFRACT24(x)                                             ((x)*0xFFFFFF)

/* Tooth Period [TCR1] and RPM */
#define RPM2TP(x)                        (TCR1_FREQ_HZ/(x)*60/(TEETH_PER_CYCLE/2))
#define TP2RPM(x)                        (TCR1_FREQ_HZ/(x)*60/(TEETH_PER_CYCLE/2))
```

- Definition of each cylinder Top Dead Center angle:

```
/* Top-Dead Centers */
#define TDC1_DEG       0
#define TDC3_DEG     180
```

```
#define TDC4_DEG      360
#define TDC2_DEG      540

/* Cam log */


#define CAM_LOG_SIZE                                                8
```

- Assignment of eTPU functions to channels:

```
/*****************************************************************************
* Define Functions to Channels
*****************************************************************************/
#define ETPU_CAM_CHAN              ETPU_ENGINE_A_CHANNEL(0)
#define ETPU_TG_CAM_CHAN           ETPU_ENGINE_A_CHANNEL(1)
#define ETPU_CRANK_CHAN            ETPU_ENGINE_A_CHANNEL(2)
#define ETPU_TG_CRANK_CHAN         ETPU_ENGINE_A_CHANNEL(3)
#define ETPU_SPARK_1_CHAN          ETPU_ENGINE_A_CHANNEL(4)
#define ETPU_SPARK_2_CHAN          ETPU_ENGINE_A_CHANNEL(5)
#define ETPU_SPARK_3_CHAN          ETPU_ENGINE_A_CHANNEL(6)
#define ETPU_SPARK_4_CHAN          ETPU_ENGINE_A_CHANNEL(7)
#define ETPU_FUEL_1_CHAN           ETPU_ENGINE_A_CHANNEL(8)
#define ETPU_FUEL_2_CHAN           ETPU_ENGINE_A_CHANNEL(9)
#define ETPU_FUEL_3_CHAN           ETPU_ENGINE_A_CHANNEL(10)
#define ETPU_FUEL_4_CHAN           ETPU_ENGINE_A_CHANNEL(11)
#define ETPU_KNOCK_1_CHAN          ETPU_ENGINE_A_CHANNEL(12)
#define ETPU_KNOCK_2_CHAN          ETPU_ENGINE_A_CHANNEL(13)
#define ETPU_INJ_BANK_1_CHAN       ETPU_ENGINE_A_CHANNEL(14)
#define ETPU_INJ_BANK_2_CHAN       ETPU_ENGINE_A_CHANNEL(15)
#define ETPU_INJ_1_CHAN            ETPU_ENGINE_A_CHANNEL(16)
#define ETPU_INJ_2_CHAN            ETPU_ENGINE_A_CHANNEL(17)
#define ETPU_INJ_3_CHAN            ETPU_ENGINE_A_CHANNEL(18)
#define ETPU_INJ_4_CHAN            ETPU_ENGINE_A_CHANNEL(19)
```

- List of channels allowed to generate an interrupt:

```
/*****************************************************************************
* Define Interrupt Enable, DMA Enable and Output Disable
*****************************************************************************/
#define ETPU_CIE_A     ( (1<<ETPU_CRANK_CHAN) \
                        +(1<<ETPU_CAM_CHAN) \
                        +(1<<ETPU_FUEL_1_CHAN) \
                        +(1<<ETPU_FUEL_2_CHAN) \
                        +(1<<ETPU_FUEL_3_CHAN) \
                        +(1<<ETPU_FUEL_4_CHAN) \
                        +(1<<ETPU_SPARK_1_CHAN) \
                        +(1<<ETPU_SPARK_2_CHAN) \
                        +(1<<ETPU_SPARK_3_CHAN) \
                        +(1<<ETPU_SPARK_4_CHAN) \
                        +(1<<ETPU_KNOCK_1_CHAN) \
                        +(1<<ETPU_KNOCK_2_CHAN) \
                        +(1<<ETPU_INJ_1_CHAN) \
                        +(1<<ETPU_INJ_2_CHAN) \
```

```
                      +(1<<ETPU_INJ_3_CHAN) \
                      +(1<<ETPU_INJ_4_CHAN) \
                      +(1<<ETPU_TG_CRANK_CHAN))
#define ETPU_DTRE_A   0x00000000
#define ETPU_ODIS_A   0x00000000
#define ETPU_OPOL_A   0x00000000
#define ETPU_CIE_B    0x00000000
#define ETPU_DTRE_B   0x00000000
#define ETPU_ODIS_B   0x00000000
#define ETPU_OPOL_B   0x00000000
```

- extern declaration of all channel parameter structures, enabling to use them. For example in *main()*:

```
/*****************************************************************************
* Global Variables Access
*****************************************************************************/
/* Global CRANK structures defined in etpu_gct.c */
extern struct crank_instance_t crank_instance;
extern struct crank_config_t   crank_config;
extern struct crank_states_t   crank_states;

/* Global CAM structures defined in etpu_gct.c */
extern struct cam_instance_t cam_instance;
extern struct cam_config_t   cam_config;
extern struct cam_states_t   cam_states;

/* Global SPARK structures defined in etpu_gct.c */
extern struct spark_instance_t spark_1_instance;
extern struct spark_instance_t spark_2_instance;
extern struct spark_instance_t spark_3_instance;
extern struct spark_instance_t spark_4_instance;
extern struct spark_config_t   spark_config;
extern struct spark_states_t   spark_1_states;
extern struct spark_states_t   spark_2_states;
extern struct spark_states_t   spark_3_states;
extern struct spark_states_t   spark_4_states;

/* Global FUEL structures defined in etpu_gct.c */
extern struct fuel_instance_t fuel_1_instance;
extern struct fuel_instance_t fuel_2_instance;
extern struct fuel_instance_t fuel_3_instance;
extern struct fuel_instance_t fuel_4_instance;
extern struct fuel_config_t   fuel_config;
extern struct fuel_states_t   fuel_1_states;
extern struct fuel_states_t   fuel_2_states;
extern struct fuel_states_t   fuel_3_states;
extern struct fuel_states_t   fuel_4_states;

/* Global INJ structures defined in etpu_gct.c */
extern struct inj_instance_t inj_1_instance;
extern struct inj_instance_t inj_2_instance;
extern struct inj_instance_t inj_3_instance;
extern struct inj_instance_t inj_4_instance;
extern struct inj_config_t   inj_config;
```

```
extern struct inj_states_t   inj_1_states;
extern struct inj_states_t   inj_2_states;
extern struct inj_states_t   inj_3_states;
extern struct inj_states_t   inj_4_states;

/* Global KNOCK structures defined in etpu_gct.c */
extern struct knock_instance_t knock_1_instance;
extern struct knock_instance_t knock_2_instance;
extern struct knock_config_t   knock_1_config;
extern struct knock_config_t   knock_2_config;

/* Global TG structures defined in etpu_gct.c */
extern struct tg_instance_t tg_instance;
extern struct tg_config_t   tg_config;
extern struct tg_states_t   tg_states;
```

- Function prototypes:

```
/*****************************************************************************
* Function Prototypes
*****************************************************************************/
int32_t my_system_etpu_init ();
void    my_system_etpu_start();
```

The ***etpu_gct.c*** file includes:

- *my_etpu_config* structure definition:
    - Note highlighted settings which configure the Enhanced Angle Counter (EAC) to use the falling transition on channel 2 input.
    - TCR1 time base is configured for full speed 100 MHz

```
/*****************************************************************************
 * Global eTPU settings - etpu_config structure
 *****************************************************************************/
/** @brief   Structure handling configuration of all global settings */

struct etpu_config_t my_etpu_config =
{
  /* etpu_config.mcr - Module Configuration Register */
  FS_ETPU_GLOBAL_TIMEBASE_DISABLE  /* keep time-bases stopped during intialization
(GTBE=0) */
  | FS_ETPU_MISC_DISABLE, /* SCM operation disabled (SCMMISEN=0) */

  /* etpu_config.misc - MISC Compare Register*/
  FS_ETPU_MISC, /* MISC compare value from etpu_set.h */

  /* etpu_config.ecr_a - Engine A Configuration Register */
  FS_ETPU_ENTRY_TABLE_ADDR /* entry table base address = shifted
FS_ETPU_ENTRY_TABLE from etpu_set.h     */
```

```
  | FS_ETPU_CHAN_FILTER_2SAMPLE /* channel filter mode = three-sample mode (CDFC=0)
*/
  | FS_ETPU_FCSS_DIV2 /* filter clock source selection = div 2 (FSCC=0) */
  | FS_ETPU_FILTER_CLOCK_DIV2 /* filter prescaler clock control = div 2 (FPSCK=0)
*/
  | FS_ETPU_PRIORITY_PASSING_ENABLE /* scheduler priority passing is enabled
(SPPDIS=0) */
  | FS_ETPU_ENGINE_ENABLE, /* engine is enabled (MDIS=0) */

  /* etpu_config.tbcr_a - Time Base Configuration Register A */
  FS_ETPU_TCRCLK_MODE_2SAMPLE /* TCRCLK signal filter control mode = two-sample
mode (TCRCF=0x) */
  | FS_ETPU_TCRCLK_INPUT_DIV2CLOCK /* TCRCLK signal filter control clock = div 2
(TCRCF=x0) */
  | FS_ETPU_TCR1CS_DIV1 /* TCR1 clock source = div 1 (TCR1CS=1)*/
  | FS_ETPU_TCR1CTL_DIV2 /* TCR1 source = div 2 (TCR1CTL=2) */
  | FS_ETPU_TCR1_PRESCALER(1) /* TCR1 prescaler = 1 (TCR1P=0) */
  | FS_ETPU_TCR2CTL_FALL /* TCR2 source = falling external (TCR2CTL=2) */
  | FS_ETPU_TCR2_PRESCALER(1) /* TCR2 prescaler = 1 (TCR2P=0) */
  | FS_ETPU_ANGLE_MODE_ENABLE_CH2, /* TCR2 angle mode is enabled, channel 2 input
(AM=3) */

  /* etpu_config.stacr_a - Shared Time And Angle Count Register A */
  FS_ETPU_TCR1_STAC_DISABLE /* TCR1 on STAC bus = disabled (REN1=0) */
  | FS_ETPU_TCR1_STAC_CLIENT /* TCR1 resource control = client (RSC1=0) */
  | FS_ETPU_TCR1_STAC_SRVSLOT(0) /* TCR1 server slot = 0 (SRV1=0) */
  | FS_ETPU_TCR2_STAC_DISABLE /* TCR2 on STAC bus = disabled (REN2=0) */
  | FS_ETPU_TCR1_STAC_CLIENT /* TCR2 resource control = client (RSC2=0) */
  | FS_ETPU_TCR2_STAC_SRVSLOT(0), /* TCR2 server slot = 0 (SRV2=0) */

  /* etpu_config.ecr_b - Engine B Configuration Register */

  /* etpu_config.tbcr_b - Time Base Configuration Register B */
  …

  /* etpu_config.stacr_b - Shared Time And Angle Count Register B */
  …

  /* etpu_config.wdtr_a - Watchdog Timer Register A(eTPU2 only) */
  FS_ETPU_WDM_DISABLED /* watchdog mode = disabled */
  | FS_ETPU_WDTR_WDCNT(0), /* watchdog count = 0 */

  /* etpu_config.wdtr_b - Watchdog Timer Register B (eTPU2 only) */
  …
};
```

- Declaration of the structures which are used as data interface to the Crank eTPU function:
  - ***Instance*** structure includes constant parameters used to initialize the eTPU function instance.
  - ***Config*** structure includes configuration parameters used for initialization and can be changed during run-time.
  - ***States*** structure reads internal states reflecting the eTPU function operation.

```
/*****************************************************************************
 * eTPU channel settings - CRANK
 *****************************************************************************/
/** @brief   Initialization of CRANK structures */
struct crank_instance_t crank_instance =
{
  ETPU_CRANK_CHAN,          /* chan_num */
  FS_ETPU_PRIORITY_HIGH,    /* priority */
  FS_ETPU_CRANK_FM0_USE_TRANS_FALLING, /* polarity */
  TEETH_TILL_GAP,           /* teeth_till_gap */
  TEETH_IN_GAP,             /* teeth_in_gap */
  TEETH_PER_CYCLE,          /* teeth_per_cycle */
  TCR2_TICKS_PER_TOOTH,     /* tcr2_ticks_per_tooth */
  FS_ETPU_CRANK_FM1_TOOTH_PERIODS_LOG_ON, /* log_tooth_periods */
  ((ETPU_CAM_CHAN <<  0) +
   (ETPU_CAM_CHAN <<  8) +
   (ETPU_CAM_CHAN << 16) +
   (ETPU_CAM_CHAN << 24)),      /* link_cam */
  ((ETPU_SPARK_1_CHAN <<  0) +
   (ETPU_SPARK_2_CHAN <<  8) +
   (ETPU_SPARK_3_CHAN << 16) +
   (ETPU_SPARK_4_CHAN << 24)), /* link_1 - sparks */
  ((ETPU_FUEL_1_CHAN <<  0) +
   (ETPU_FUEL_2_CHAN <<  8) +
   (ETPU_FUEL_3_CHAN << 16) +
   (ETPU_FUEL_4_CHAN << 24)),  /* link_2 - fuels */
  ((ETPU_KNOCK_1_CHAN <<  0) +
   (ETPU_KNOCK_2_CHAN <<  8) +
   (ETPU_INJ_BANK_1_CHAN << 16) +
   (ETPU_INJ_BANK_2_CHAN << 24)),/* link_3 - knocks and inj_banks */
  ((ETPU_INJ_1_CHAN <<  0) +
   (ETPU_INJ_2_CHAN <<  8) +
   (ETPU_INJ_3_CHAN << 16) +
   (ETPU_INJ_4_CHAN << 24)),    /* link_4 - injs */
  0,                        /* *cpba */  /* 0 for automatic allocation */
  0                         /* *cpba_tooth_period_log */  /* automatic allocation */
};

struct crank_config_t crank_config =
{
  1*(TEETH_TILL_GAP+TEETH_IN_GAP), /* teeth_per_sync */
  MSEC2TCR1(10), /* blank_time */
  5,             /* blank_teeth */
  UFRACT24(0.6), /* gap_ratio */
  UFRACT24(0.2), /* win_ratio_normal */
  UFRACT24(0.5), /* win_ratio_across_gap */
  UFRACT24(0.2), /* win_ratio_after_gap */
  UFRACT24(0.5), /* win_ratio_after_timeout */
  MSEC2TCR1(50)  /* first_tooth_timeout */
};

struct crank_states_t crank_states;
```

- Declaration of structures which are used as a data interface to the Cam eTPU function:

```
/*****************************************************************************
 * eTPU channel settings - CAM
 *****************************************************************************/
/** @brief   Initialization of CAM structures */
struct cam_instance_t cam_instance =
{
  ETPU_CAM_CHAN,          /* chan_num */
  FS_ETPU_PRIORITY_LOW,   /* priority */
  CAM_LOG_SIZE,           /* log_size */
  0,                      /* *cpba */     /* 0 for automatic allocation */
  0                       /* *cpba_log */ /* 0 for automatic allocation */
};

struct cam_config_t cam_config =
{
  FS_ETPU_CAM_LOG_BOTH /* mode */
};

struct cam_states_t cam_states;
```

- Declaration of structures which are used as data interface to four instances of the Spark eTPU function:
    - All instances share the same *Config* structure because similar spark parameters are used on all of the four cylinders.
    - There are two items defined (two sparks 360º apart) in the array of single sparks, where only the first one is used.

```
/*****************************************************************************
 * eTPU channel settings - SPARKs
 *****************************************************************************/
/** @brief   Initialization of SPARK structures */
struct spark_instance_t spark_1_instance =
{
  ETPU_SPARK_1_CHAN,        /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE,  /* priority */
  FS_ETPU_SPARK_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC1_DEG),       /* tdc_angle */
0,                        /* *cpba */               /* 0 for automatic allocation */
  0                         /* *cpba_single_spark */  /* 0 for automatic allocation
*/
};

struct spark_instance_t spark_2_instance =
{
  ETPU_SPARK_2_CHAN,        /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE,  /* priority */
  FS_ETPU_SPARK_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC2_DEG),       /* tdc_angle */
```

```
  0,                        /* *cpba */               /* 0 for automatic allocation
*/
  0                         /* *cpba_single_spark */  /* 0 for automatic allocation
*/
};

struct spark_instance_t spark_3_instance =
{
  ETPU_SPARK_3_CHAN,        /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE, /* priority */
  FS_ETPU_SPARK_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC3_DEG),       /* tdc_angle */
  0,                        /* *cpba */               /* 0 for automatic allocation
*/
  0                         /* *cpba_single_spark */  /* 0 for automatic allocation
*/
};

struct spark_instance_t spark_4_instance =
{
  ETPU_SPARK_4_CHAN,        /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE, /* priority */
  FS_ETPU_SPARK_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC4_DEG),       /* tdc_angle */
  0,                        /* *cpba */               /* 0 for automatic allocation
*/
  0                         /* *cpba_single_spark */  /* 0 for automatic allocation
*/
};

struct single_spark_config_t single_spark_config[2] =
{
  {
    DEG2TCR2(0),          /* end_angle */
    USEC2TCR1(2000),      /* dwell_time */
    3                     /* multi_pulse_count */
  },
  {
    DEG2TCR2(-360),       /* end_angle */
    USEC2TCR1(2000),      /* dwell_time */
    3                     /* multi_pulse_count */
  }
};

struct spark_config_t spark_config =
{
  DEG2TCR2(30),     /* angle_offset_recalc */
  USEC2TCR1(1800), /* dwell_time_min */
  USEC2TCR1(2200), /* dwell_time_max */
  USEC2TCR1(100),  /* multi_on_time */
  USEC2TCR1(100),  /* multi_off_time */
  1,                /* spark_count */
  &single_spark_config[0],  /* p_single_spark_config */
  FS_ETPU_SPARK_GENERATION_ALLOWED  /* generation_disable */
};
```

```
struct spark_states_t spark_1_states;
struct spark_states_t spark_2_states;
struct spark_states_t spark_3_states;
struct spark_states_t spark_4_states;
```

- Declaration of structures which are used as data interface to four instances of the Fuel eTPU function (all instances share the same *Config* structure because similar fuel parameters are used on all of the four cylinders):

```
/*****************************************************************************
 * eTPU channel settings - FUELs
 *****************************************************************************/
/** @brief   Initialization of FUEL structures */
struct fuel_instance_t fuel_1_instance =
{
  ETPU_FUEL_1_CHAN,          /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE, /* priority */
  FS_ETPU_FUEL_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC1_DEG),       /* tdc_angle */
  0                         /* *cpba */  /* 0 for automatic allocation */
};

struct fuel_instance_t fuel_2_instance =
{
  ETPU_FUEL_2_CHAN,          /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE, /* priority */
  FS_ETPU_FUEL_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC2_DEG),       /* tdc_angle */
  0                         /* *cpba */  /* 0 for automatic allocation */
};

struct fuel_instance_t fuel_3_instance =
{
  ETPU_FUEL_3_CHAN,          /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE, /* priority */
  FS_ETPU_FUEL_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC3_DEG),       /* tdc_angle */
  0                         /* *cpba */  /* 0 for automatic allocation */
};

struct fuel_instance_t fuel_4_instance =
{
  ETPU_FUEL_4_CHAN,          /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE, /* priority */
  FS_ETPU_FUEL_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC4_DEG),       /* tdc_angle */
  0                         /* *cpba */  /* 0 for automatic allocation */
};

struct fuel_config_t fuel_config =
{
  DEG2TCR2(60),      /* angle_normal_end */
  DEG2TCR2(40),      /* angle_stop */
```

```
  DEG2TCR2(30),      /* angle_offset_recalc */
  USEC2TCR1(20000), /* injection_time */
  USEC2TCR1(1000),  /* compensation_time */
  USEC2TCR1(1000),  /* injection_time_minimum */
  USEC2TCR1(1000)   /* off_time_minimum */
};
struct fuel_states_t fuel_1_states;
struct fuel_states_t fuel_2_states;
struct fuel_states_t fuel_3_states;
struct fuel_states_t fuel_4_states;
```

- Declaration of structures which are used as a data interface to four instances of the Direct Injection eTPU function:
    - All cylinders share the same definition of injection sequence, consisting of three injections, where each injection is defined by five, seven, or three phases.

```
/*****************************************************************************
 * eTPU channel settings - INJ
 *****************************************************************************/
/** @brief   Initialization of INJ structures */
struct inj_instance_t inj_1_instance =
{
  ETPU_INJ_1_CHAN,        /* chan_num_inj */
  ETPU_INJ_BANK_1_CHAN,  /* chan_num_bank_1 */
  ETPU_INJ_BANK_2_CHAN,  /* chan_num_bank_2 */
  FS_ETPU_INJ_BANK_CHAN_NOT_USED, /* chan_num_bank_3 */
  FS_ETPU_PRIORITY_HIGH,  /* priority */
  FS_ETPU_INJ_FM0_ACTIVE_HIGH, /* polarity_inj */
  FS_ETPU_INJ_FM0_ACTIVE_HIGH, /* polarity_bank */
  DEG2TCR2(TDC1_DEG),     /* tdc_angle */
  0,                      /* *cpba */  /* 0 for automatic allocation */
  0,                      /* *cpba_injections */
  0                       /* *cpba_phases */
};
struct inj_instance_t inj_2_instance =
{
  ETPU_INJ_2_CHAN,        /* chan_num_inj */
  ETPU_INJ_BANK_1_CHAN,  /* chan_num_bank_1 */
  ETPU_INJ_BANK_2_CHAN,  /* chan_num_bank_2 */
  FS_ETPU_INJ_BANK_CHAN_NOT_USED, /* chan_num_bank_3 */
  FS_ETPU_PRIORITY_HIGH,  /* priority */
  FS_ETPU_INJ_FM0_ACTIVE_HIGH, /* polarity_inj */
  FS_ETPU_INJ_FM0_ACTIVE_HIGH, /* polarity_bank */
  DEG2TCR2(TDC2_DEG),     /* tdc_angle */
  0,                      /* *cpba */  /* 0 for automatic allocation */
  0,                      /* *cpba_injections */
  0                       /* *cpba_phases */
};
struct inj_instance_t inj_3_instance =
{
  ETPU_INJ_3_CHAN,        /* chan_num_inj */
  ETPU_INJ_BANK_1_CHAN,  /* chan_num_bank_1 */
  ETPU_INJ_BANK_2_CHAN,  /* chan_num_bank_2 */
```

```
  FS_ETPU_INJ_BANK_CHAN_NOT_USED, /* chan_num_bank_3 */
  FS_ETPU_PRIORITY_HIGH,  /* priority */
  FS_ETPU_INJ_FM0_ACTIVE_HIGH, /* polarity_inj */
  FS_ETPU_INJ_FM0_ACTIVE_HIGH, /* polarity_bank */
  DEG2TCR2(TDC3_DEG),     /* tdc_angle */
  0,                      /* *cpba */  /* 0 for automatic allocation */
  0,                      /* *cpba_injections */
  0                       /* *cpba_phases */
};
struct inj_instance_t inj_4_instance =
{
  ETPU_INJ_4_CHAN,        /* chan_num_inj */
  ETPU_INJ_BANK_1_CHAN,   /* chan_num_bank_1 */
  ETPU_INJ_BANK_2_CHAN,   /* chan_num_bank_2 */
  FS_ETPU_INJ_BANK_CHAN_NOT_USED, /* chan_num_bank_3 */
  FS_ETPU_PRIORITY_HIGH,  /* priority */
  FS_ETPU_INJ_FM0_ACTIVE_HIGH, /* polarity_inj */
  FS_ETPU_INJ_FM0_ACTIVE_HIGH, /* polarity_bank */
  DEG2TCR2(TDC4_DEG),     /* tdc_angle */
  0,                      /* *cpba */  /* 0 for automatic allocation */
  0,                      /* *cpba_injections */
  0                       /* *cpba_phases */
};

uint32_t inj_injection_1_phase_config[5] =
{
/*     duration,  BANK 1 output,                       BANK 2 output,
INJ output */
  USEC2TCR1(20) + FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_1 + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(10) + FS_ETPU_INJ_PHASE_OUT_LOW         + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(30) + FS_ETPU_INJ_PHASE_OUT_LOW         +
FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_2 + FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(10) + FS_ETPU_INJ_PHASE_OUT_LOW         + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(50) + FS_ETPU_INJ_PHASE_OUT_LOW         +
FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_2 + FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
};
uint32_t inj_injection_2_phase_config[7] =
{
/*     duration,  BANK 1 output,                       BANK 2 output,
INJ output */
  USEC2TCR1(20) + FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_1 + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(10) + FS_ETPU_INJ_PHASE_OUT_LOW         + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(30) + FS_ETPU_INJ_PHASE_OUT_LOW         +
FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_2 + FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(10) + FS_ETPU_INJ_PHASE_OUT_LOW         + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(100)+ FS_ETPU_INJ_PHASE_OUT_LOW         +
FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_2 + FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1( 5) + FS_ETPU_INJ_PHASE_OUT_LOW         + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
```

```
  USEC2TCR1(50) + FS_ETPU_INJ_PHASE_OUT_LOW         +
FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_2 + FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
};
uint32_t inj_injection_3_phase_config[3] =
{
/*     duration,  BANK 1 output,                      BANK 2 output,
INJ output */
  USEC2TCR1(20) + FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_1 + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(10) + FS_ETPU_INJ_PHASE_OUT_LOW         + FS_ETPU_INJ_PHASE_OUT_LOW
+ FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
  USEC2TCR1(40) + FS_ETPU_INJ_PHASE_OUT_LOW         +
FS_ETPU_INJ_PHASE_OUT_HIGH_BANK_2 + FS_ETPU_INJ_PHASE_OUT_HIGH_INJ,
};

struct inj_injection_config_t inj_injection_config[3] =
{
  {
    DEG2TCR2(20),         /* angle_start */
    5,                    /* phase_count */
    &inj_injection_1_phase_config[0]  /* *p_phase_config */
  },
  {
    DEG2TCR2(10),         /* angle_start */
    7,                    /* phase_count */
    &inj_injection_2_phase_config[0]  /* *p_phase_config */
  },
  {
    DEG2TCR2(-5),         /* angle_start */
    3,                    /* phase_count */
    &inj_injection_3_phase_config[0]  /* *p_phase_config */
  }
};

struct inj_config_t inj_config =
{
  DEG2TCR2(90),             /* angle_irq */
DEG2TCR2(-20),           /* angle_stop */
  3,                        /* injection_count */
  &inj_injection_config[0] /* *p_inj_injection_config */
};

struct inj_states_t inj_1_states;
struct inj_states_t inj_2_states;
struct inj_states_t inj_3_states;
struct inj_states_t inj_4_states;
```

- Declaration of structures which are used as data interface for two instances of the Knock eTPU function:
    - Each Knock channel generates two windows. The first one starts 90º before the TDC (see *`DEG2TCR2(90), /* angle_start */`*) and the second one 360º later (see *`DEG2TCR2(90-360), /* angle_start */`*).

```
/*****************************************************************************
 * eTPU channel settings - KNOCKs
 *****************************************************************************/
/** @brief   Initialization of KNOCK structures */
struct knock_instance_t knock_1_instance =
{
  ETPU_KNOCK_1_CHAN,       /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE, /* priority */
  FS_ETPU_KNOCK_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC1_DEG),      /* tdc_angle */
  0,                       /* *cpba */          /* 0 for automatic allocation */
  0                        /* *cpba_windows */  /* 0 for automatic allocation */
};

struct knock_instance_t knock_2_instance =
{
  ETPU_KNOCK_2_CHAN,       /* chan_num */
  FS_ETPU_PRIORITY_MIDDLE, /* priority */
  FS_ETPU_KNOCK_FM0_ACTIVE_HIGH, /* polarity */
  DEG2TCR2(TDC3_DEG),      /* tdc_angle */
  0,                       /* *cpba */          /* 0 for automatic allocation */
  0                        /* *cpba_windows */  /* 0 for automatic allocation */
};

struct knock_window_config_t knock_window_config[2] =
{
  {
    DEG2TCR2(90),      /* angle_start */
    DEG2TCR2(180)      /* angle_width */
  },
  {
    DEG2TCR2(90-360), /* angle_start */
    DEG2TCR2(180)      /* angle_width */
  }
};

struct knock_config_t knock_1_config =
{
  FS_ETPU_KNOCK_FM1_MODE_TRIGGER,    /* mode */
  2,                                 /* window_count */
  &knock_window_config[0],           /* p_knock_window_config */
USEC2TCR1(100),                    /* trigger_period */
  FS_ETPU_KNOCK_IRQ_AT_WINDOW_END    /* irq_dma_options */
};

struct knock_config_t knock_2_config =
{
  FS_ETPU_KNOCK_FM1_MODE_TRIGGER,    /* mode */
  2,                                 /* window_count */
  &knock_window_config[0],           /* p_knock_window_config */
  USEC2TCR1(100),                    /* trigger_period */
  FS_ETPU_KNOCK_IRQ_AT_WINDOW_END    /* irq_dma_options */
};
```

- Declaration of structures which are used as data interface to Tooth Generator eTPU function:
    - The *cam_edge_teeth* array defines the generated Cam pattern

```
/*****************************************************************************
 * eTPU channel settings - TG
 *****************************************************************************/
/** @brief   Initialization of TG structures */
uint8_t cam_edge_teeth[] = {
  6, 12, 27,
  36+15, 36+24, 36+30
};

struct tg_instance_t tg_instance =
{
  ETPU_TG_CRANK_CHAN,     /* chan_num_crank */
  ETPU_TG_CAM_CHAN,       /* chan_num_cam */
  FS_ETPU_PRIORITY_LOW,   /* priority */
  FS_ETPU_TG_FM0_POLARITY_LOW, /* polarity_crank */
  FS_ETPU_TG_FM0_POLARITY_LOW, /* polarity_cam */
  TEETH_TILL_GAP,         /* teeth_till_gap */
  TEETH_IN_GAP,           /* teeth_in_gap */
  TEETH_PER_CYCLE,        /* teeth_per_cycle */
  sizeof(cam_edge_teeth),/* cam_edge_count */
  &cam_edge_teeth[0],     /* *p_cam_edge_tooth */
  0,                      /* *cpba */  /* 0 for automatic allocation */
  0                       /* *cpba_cam_edge_tooth */
};

struct tg_config_t tg_config =
{
  RPM2TP(5000),     /* tooth_period_target */
  UFRACT24(0.1),  /* accel_rate */
  FS_ETPU_TG_GENERATION_ALLOWED /* generation_disable */
};

struct tg_states_t tg_states;
```

- The TSA tables which enables FreeMASTER to access the eTPU data structures:

```
/*****************************************************************************
* FreeMASTER TSA tables
*****************************************************************************/
…
```

- The function *my_system_etpu_init()* which initializes global eTPU settings first and then initializes each eTPU channel, using API initialization routines and the data structures declared above:

```
/*****************************************************************************
* FUNCTION: my_system_etpu_init
*****************************************************************************//*!
```

```
* @brief   This function initialize the eTPU module:
*          -# Initialize global setting using fs_etpu_init function
*             and the my_etpu_config structure
*          -# Initialize channel setting using channel function APIs
*
* @return  Zero or an error code is returned.
*******************************************************************************/
int32_t my_system_etpu_init()
{
  int32_t err_code;

  /* Initialization of eTPU global settings */
  err_code = fs_etpu_init(
    my_etpu_config,
    (uint32_t *)etpu_code, sizeof(etpu_code),
    (uint32_t *)etpu_globals, sizeof(etpu_globals));
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code);

  /* Initialization of eTPU channel settings */
  err_code = fs_etpu_crank_init(
    &crank_instance,
    &crank_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_CRANK_CHAN<<16));

  err_code = fs_etpu_cam_init(
    &cam_instance,
    &cam_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_CAM_CHAN<<16));

  err_code = fs_etpu_spark_init(
    &spark_1_instance,
    &spark_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_SPARK_1_CHAN<<16));

 err_code = fs_etpu_spark_init(
    &spark_2_instance,
    &spark_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_SPARK_2_CHAN<<16));

  err_code = fs_etpu_spark_init(
    &spark_3_instance,
    &spark_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_SPARK_3_CHAN<<16));

  err_code = fs_etpu_spark_init(
    &spark_4_instance,
    &spark_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_SPARK_4_CHAN<<16));

  err_code = fs_etpu_fuel_init(
    &fuel_1_instance,
    &fuel_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_FUEL_1_CHAN<<16));

  …
```

```
  err_code = fs_etpu_inj_init(
    &inj_1_instance,
    &inj_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_INJ_1_CHAN<<16));

  …

  err_code = fs_etpu_knock_init(
    &knock_1_instance,
    &knock_1_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_KNOCK_1_CHAN<<16));

  err_code = fs_etpu_knock_init(
    &knock_2_instance,
    &knock_2_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_KNOCK_2_CHAN<<16));

  err_code = fs_etpu_tg_init(
    &tg_instance,
    &tg_config);
  if(err_code != FS_ETPU_ERROR_NONE) return(err_code + (ETPU_TG_CRANK_CHAN<<16));

  return(FS_ETPU_ERROR_NONE);
}
```

- The function *my_system_etpu_start()* which applies the masks enabling the channel interrupts, DMA requests (N/A), and Output Disable (N/A) before it synchronously starts all eTPU clocks:

```
/*****************************************************************************
* FUNCTION: my_system_etpu_start
*****************************************************************************//*!
* @brief   This function enables channel interrupts, DMA requests and "output
*          disable" feature on selected channels and starts TCR time bases using
*          Global Timebase Enable (GTBE) bit.
* @warning This function should be called after all device modules, including
*          the interrupt and DMA controller, are configured.
*****************************************************************************/
void my_system_etpu_start()
{
  /* Initialization of Interrupt Enable, DMA Enable
     and Output Disable channel options */
  fs_etpu_set_interrupt_mask_a(ETPU_CIE_A);
  fs_etpu_set_interrupt_mask_b(ETPU_CIE_B);
  fs_etpu_set_dma_mask_a(ETPU_DTRE_A);
  fs_etpu_set_dma_mask_b(ETPU_DTRE_B);
  fs_etpu_set_output_disable_mask_a(ETPU_ODIS_A, ETPU_OPOL_A);
  fs_etpu_set_output_disable_mask_b(ETPU_ODIS_B, ETPU_OPOL_B);

  /* Synchronous start of all TCR time bases */
  fs_timer_start();
}
```

Finally, both initialization routines are called from *main()*:

- *my_system_etpu_init()* is called as one of the module initialization routines.
- *my_system_etpu_start()* is called after all modules are initialized, interrupts are enabled, and the application can start running.

3.2. eTPU run-time interface

If the CPU application does not interface eTPU during run-time, the eTPU continues to drive the engine using the last parameter values (for example, fuel injection time). The CPU can only interface eTPU in order to monitor the operation (for example read engine speed) or set new parameter values.

In this demo application, the eTPU functions are interfaced in eTPU channel interrupt handlers. The channel interrupts are generated at specific moments as listed in the table below.

Table 5 Engine control eTPU function channel interrupts

eTPU function	Channel interrupt condition
Crank	The *eng_pos_state* is changed. On the first tooth during synchronization, when the Cam log is buffered and is ready for recognition. In full synchronization state, once per engine cycle, on the first tooth.
Cam	On detecting an error condition
Spark	Before each single spark, on *recalc_angle*
Fuel	Every engine cycle, at *angle_stop*
Inj	Before each injection sequence, at *angle_irq*
Knock	Selectively at: window start window end (used) every trigger pulse
Tooth Generator	On every gap

All these channel interrupts are used to interface the eTPU functions. The following Tooth Generator channel interrupt handler code can serve as an example:

```
/*************************************************************************//*!
*
* @brief   Interrupt from eTPU channel TG
*
* @return  N/A
*
* @note    This interrupt is generated in each gap. The Tooth Generator
*          parameters can be adjusted.
*
*************************************************************************/
void etpu_tg_isr(void)
```

```
{
  fs_gpio_write_data(TEST_PAD_TG, 1);

  fs_etpu_clear_chan_interrupt_flag(ETPU_TG_CRANK_CHAN);

  /* Interface TG eTPU function */
  fs_etpu_tg_get_states(&tg_instance, &tg_states);
  fs_etpu_tg_config(&tg_instance, &tg_config);

  fs_gpio_write_data(TEST_PAD_TG, 0);
}
```

The eTPU function is interfaced using the API routines. The eTPU function internal states are read out from the eTPU function to the *States* structure using the *fs_etpu_tg_get_states()* function. The eTPU function parameters are updated by the latest values from the *Config* structure using *fs_etpu_tg_config()* function. This demo application does not implement any software to manipulate this data. Instead, the data is accessible from FreeMASTER. The *States* structure data is read-only and is just visualized in FreeMASTER GUI, while the *Config* structure data can be modified from FreeMASTER.

In order to check the interrupt timing using a scope, each interrupt service routine toggle a Test Pad high at the beginning and low at the end.

All other eTPU function channel interrupts are serviced similarly.

Additionally, in order to enable the smooth behavior of the FreeMASTER Control Page GUI, some data is exchanged at a faster rate, in the background loop:

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

- Fuel injection time is set

```
  /* Set Fuel injection time - the value is updated in by FreeMASTER */
  fs_etpu_fuel_update_injection_time(&fuel_1_instance, &fuel_config);
```

- Tooth Generator configuration is written in and states read out

```
  /* Interface TG eTPU function - this sets engine speed updated by FreeMASTER */
  fs_etpu_tg_get_states(&tg_instance, &tg_states);
  fs_etpu_tg_config(&tg_instance, &tg_config);
```

- Cam and Crank states are read out

```
  /* update Crank and Cam latest states to see them in FreeMaster*/
  fs_etpu_crank_get_states(&crank_instance, &crank_states);
  fs_etpu_cam_get_states(&cam_instance, &cam_states);

  /* FreeMASTER processing on background */
  FMSTR_Poll();
}
```

3.2.1. Crank channel interrupt handler

The Crank eTPU function is the only one where a proper interrupt handler is mandatory for the engine control operation. The CPU application needs to assist eTPU during engine synchronization process. The task of CPU is to recognize the Cam pattern logged by the eTPU function Cam as illustrated in Figure 3.

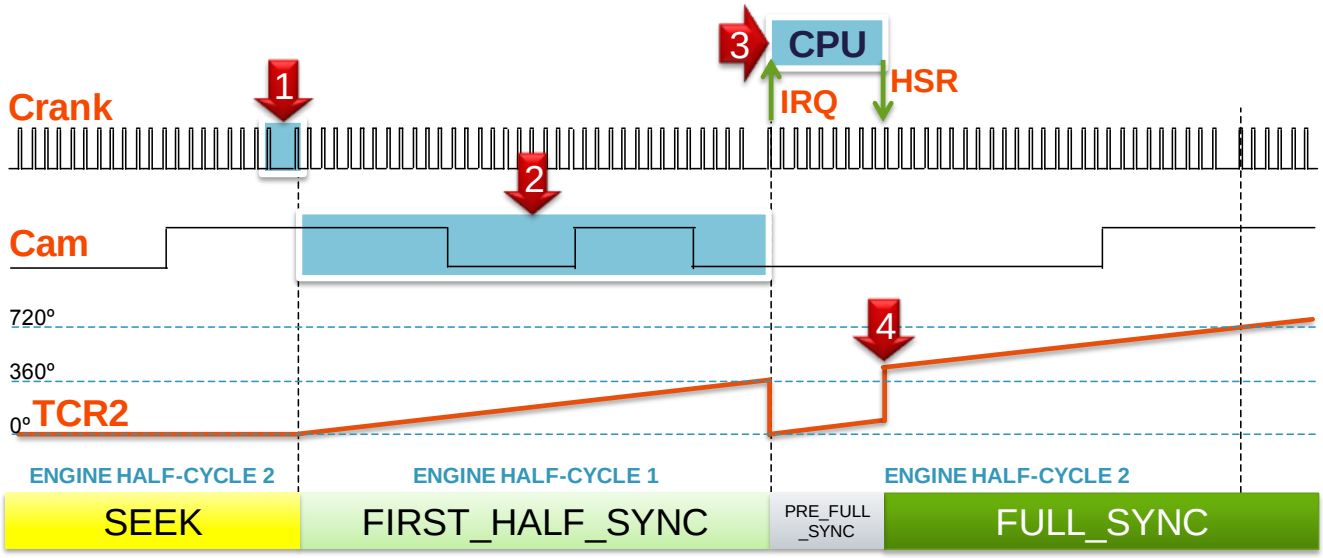


Figure 3 Engine synchronization example

1. eTPU function Crank recognizes the gap or the additional tooth on the crank wheel.
2. eTPU function Cam logs Cam signal transitions, after which Crank generates a channel interrupt.
3. On the Crank interrupt, CPU uses Cam log to decode the engine position. The CPU writes the decoded TCR2 engine angle (corresponding to the first tooth after gap) to Crank eTPU function, using the API function *fs_etpu_crank_set_sync()*.
4. eTPU function Crank adjusts the engine angle TCR2 value and hence full synchronization is achieved.

The Crank channel interrupt is generated not only during the synchronization process, but also in other cases. Because of this, the interrupt handler needs to use the *eng_pos_state* variable value to distinguish between various cases:

```
switch(crank_states.eng_pos_state)
{
case FS_ETPU_ENG_POS_SEEK:
  /* Crank has stalled */
  break;
case FS_ETPU_ENG_POS_FIRST_HALF_SYNC:
```

```
    /* Crank has found the gap */
    break;
  case FS_ETPU_ENG_POS_PRE_FULL_SYNC:
    /* Cam signal is logged.
       NOW THE CPU MUST RECOGNIZE THE LOGGED CAM PATTERN */
    break;
  case FS_ETPU_ENG_POS_FULL_SYNC:
    /* Regular interrupt on the first tooth every engine cycle. */
    break;
  }
```

Cam pattern recognition is expected from the CPU application only in the Pre-Full Sync mode (that is, when *eng_pos_state* is equal to *FS_ETPU_ENG_POS_PRE_FULL_SYNC*). This demo application implements a very simple example of the recognition.

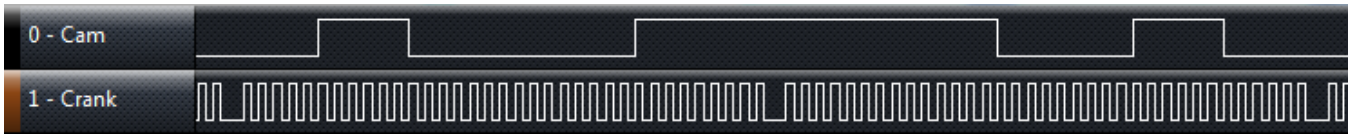


Figure 4 Cam pattern

The Cam pattern consists of six transitions per engine cycle. There are three transitions in the first half-cycle and three in the second half-cycle. The transition polarities are different in each half-cycle. The log of the first half-cycle starts by a rising transition while the log of the second half-cycle starts by a falling transition. The simple Cam pattern recognition algorithm checks the number of transitions logged and the first transition polarity, but does not check the transition angles.

```
fs_etpu_cam_get_states(&cam_instance, &cam_states);
 fs_etpu_cam_copy_log(&cam_instance, &etpu_cam_log[0]);

 if((cam_states.log_idx == 3) && (etpu_cam_log[0] & 0x01000000))
 {
   /* 3 transition logged,
      the first transition is a rising one
      => the first half-cycle was logged */
   /* Set, what angle should have been at the last gap */
   tcr2_adjustment = DEG2TCR2(360);
 }
 else if((cam_states.log_idx == 3) && ~(etpu_cam_log[0] & 0x01000000))
 {
   /* 3 transitions logged,
      the first transition is a falling one
      => the second half-cycle was logged */
   /* Set, what angle should have been at the last gap */
   tcr2_adjustment = DEG2TCR2(0);
 }
 else
 {
   /* Cam pattern is not recognized */
   break;
 }
 fs_etpu_crank_set_sync(&crank_instance, tcr2_adjustment);
```

4. Screenshots

4.1. FreeMASTER GUI on PC

FreeMASTER tool enables the control and monitoring of the application running on MPC5674F. FreeMASTER can access MPC5674F memory to read or write application variables via the serial RS232 connection. This allows all of the data structures used to interface the eTPU functions to be available on the PC screen. The user can read the variables and modify some of them in the watch pane. Additionally, the graphical control page enables the start and stop of the tooth generator, sets the engine speed using a graphical gauge, and set the injection time in range 0 to 50 ms by the Throttle gauge. Besides that, the control page enables to monitor the synchronization states and all error flags via small light indicators.

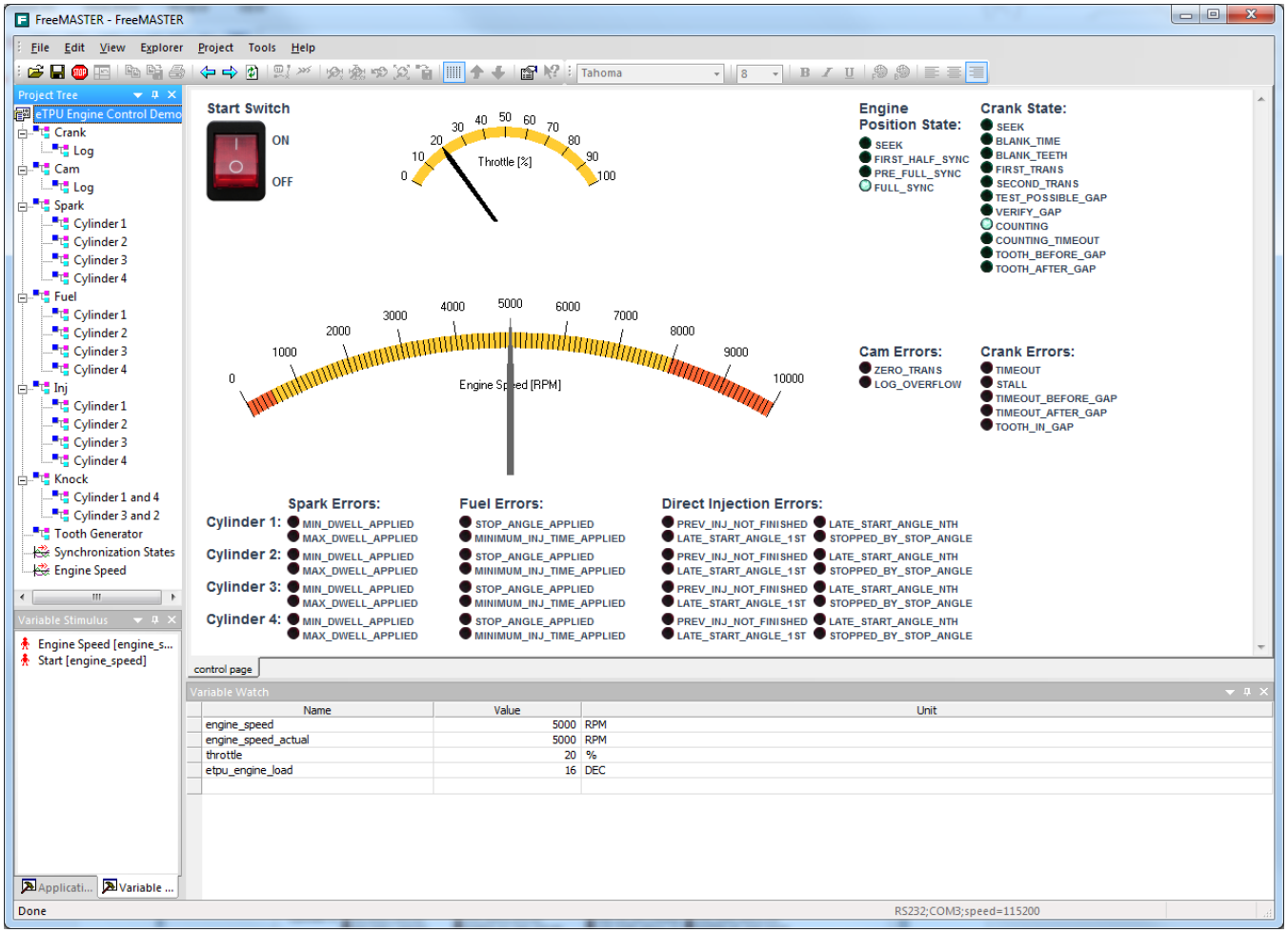


Figure 5 FreeMASTER control page

The FreeMASTER also enables to see life plots of selected variable values. 0 shows various state variables during the synchronization process.

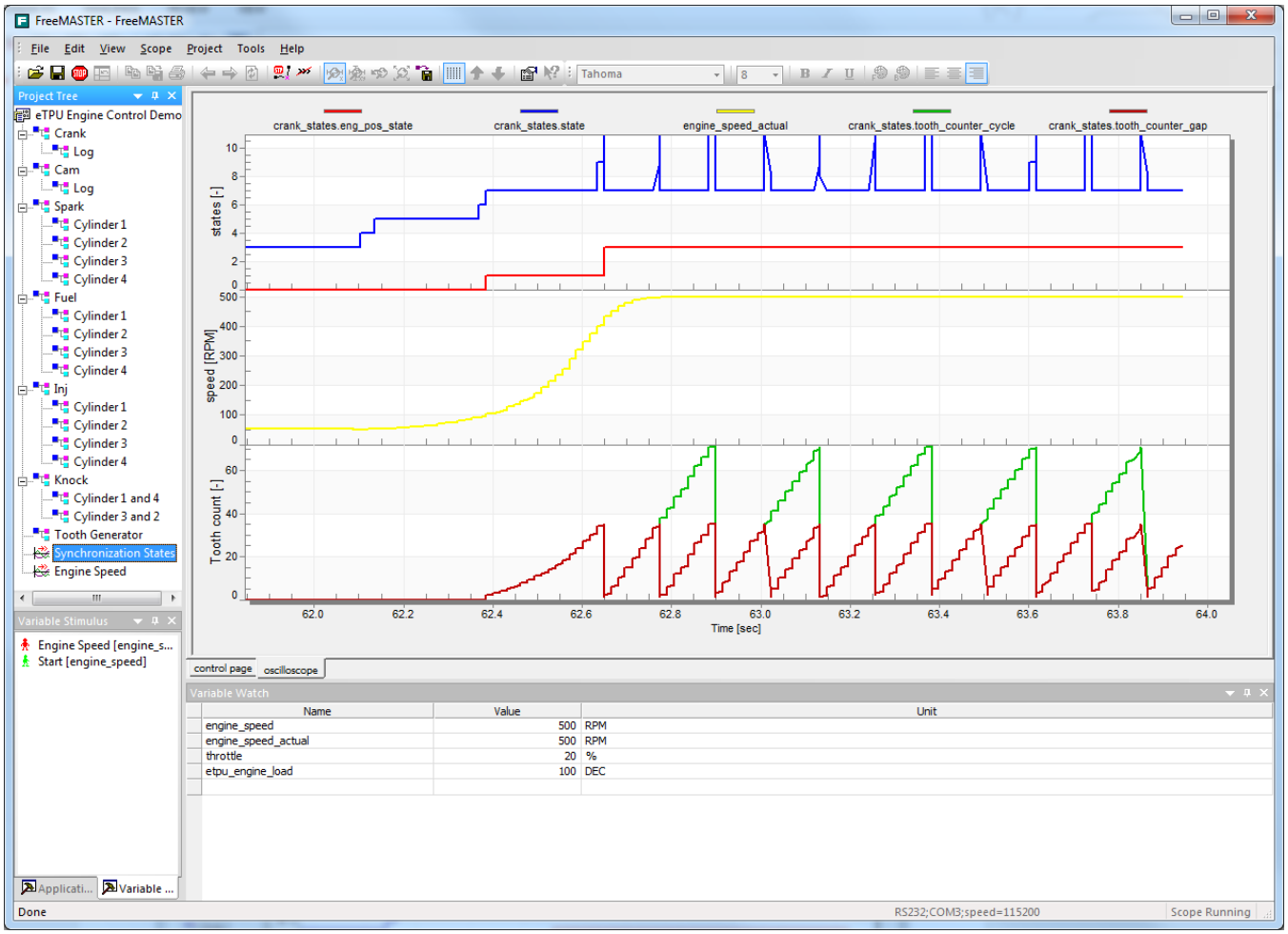


Figure 6 FreeMASTER life plot of synchronization process

4.2. Generated signals

The logic analyzer screenshots depict the demo application input and output signals on the following figures. The IRQ signal at the bottom is measured on one of the test pads and shows an activity of particular interrupt handlers. For all of the figures, the engine speed is 5000 rpm.

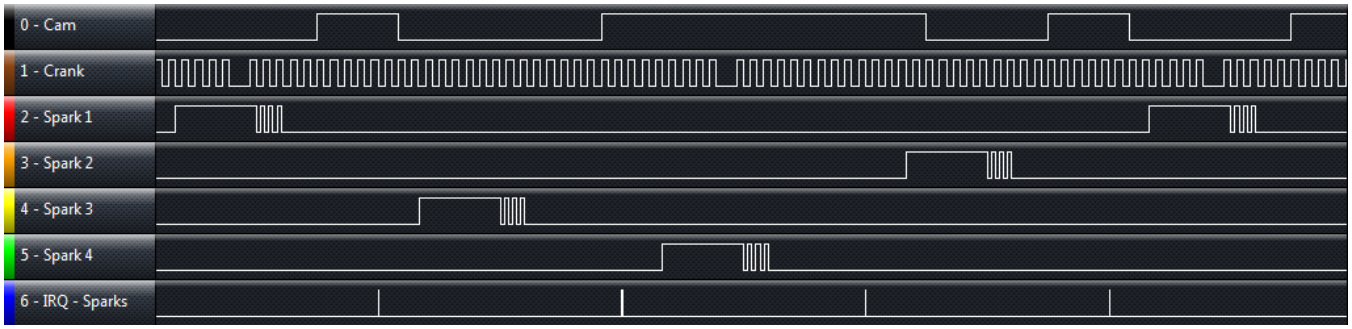


Figure 7 Cam and Crank inputs with Spark outputs (2 ms dwell-time and 3 multipulses)

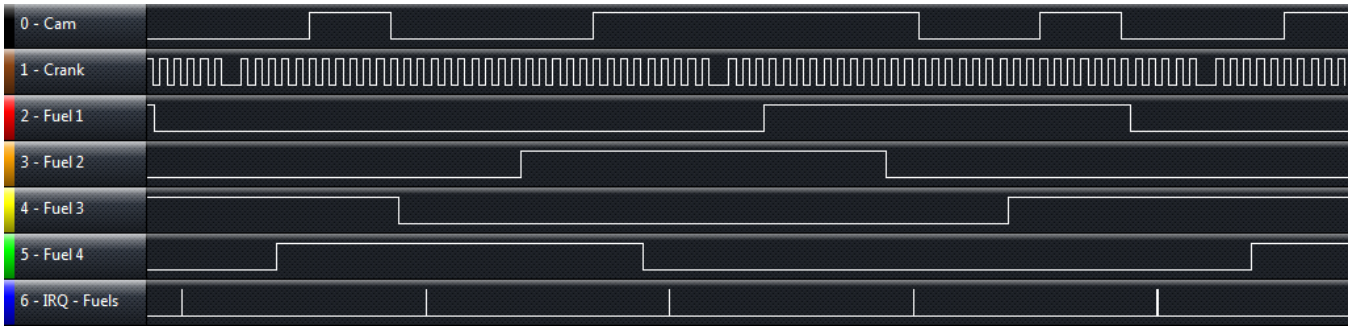


Figure 8 Cam and Crank inputs with Fuel outputs (8 ms injection time and 1 ms compensation time)

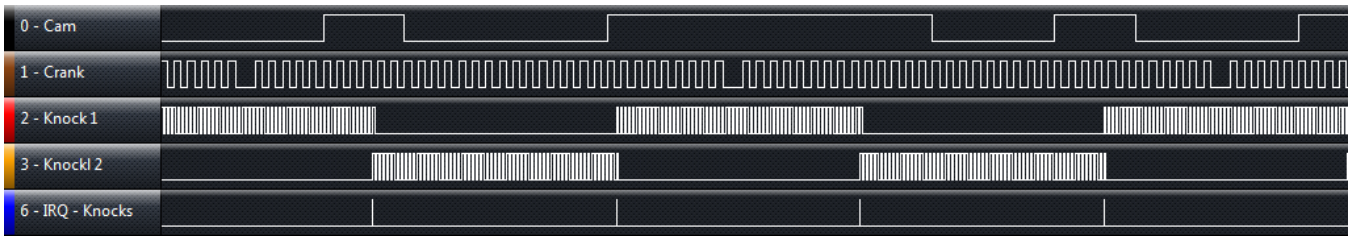


Figure 9 Cam and Crank inputs with Knock outputs (180° angle windows, trigger mode)

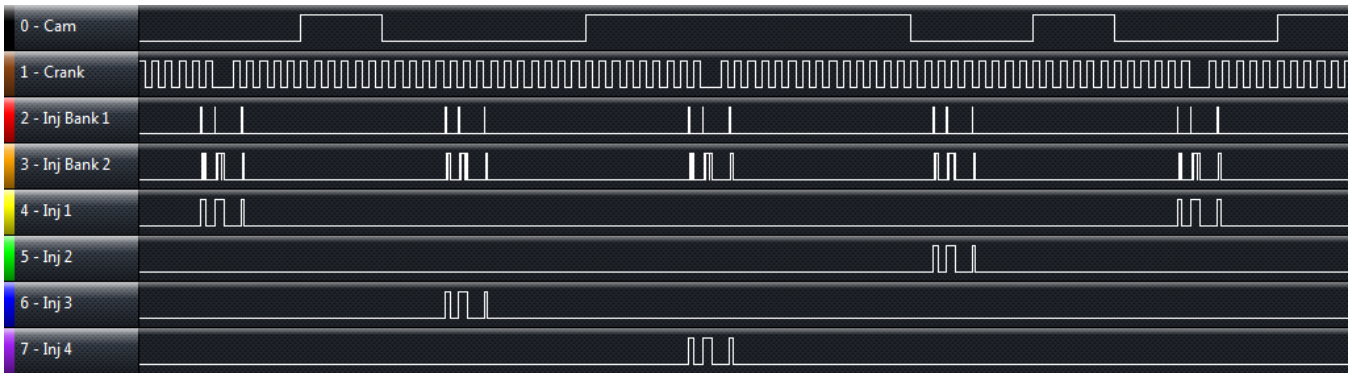


Figure 10 Cam and Crank inputs with Direct Injection outputs (2 bank channels, 3 injections in each sequence)

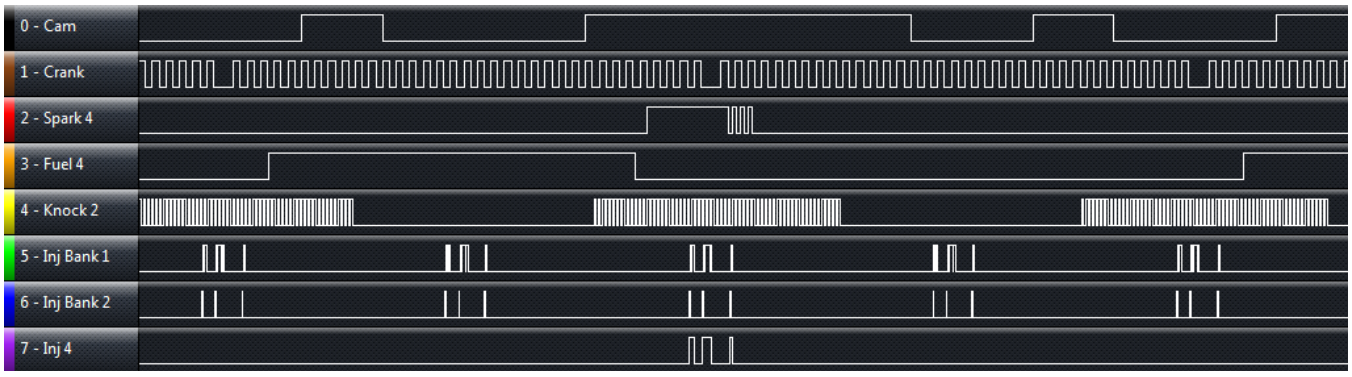


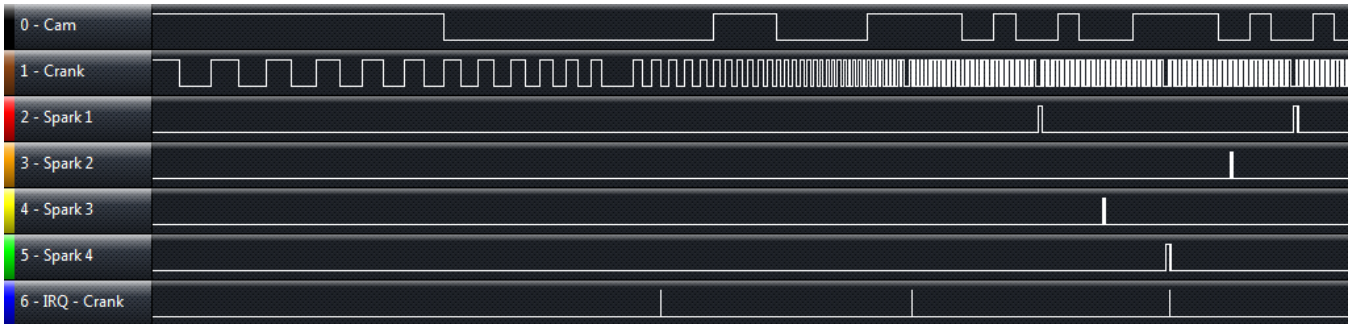
Figure 11 Cam and Crank inputs with Spark, Fuel, Knock and Direct Injection outputs, all targeted to cylinder 4

Figure 11 shows the timing of the engine start. The bottom signal includes three pulses, to show the activity of the Crank channel interrupt handler:

1st pulse: The first gap was found (*FS_ETPU_ENG_POS_FIRST_HALF_SYNC*).

2nd pulse: The logged Cam pattern is recognized (*FS_ETPU_ENG_POS_PRE_FULL_SYNC*).

3rd pulse: The first of the regular interrupts generated each engine cycle (*FS_ETPU_ENG_POS_FULL_SYNC*).



5. Summary

The new Engine Control eTPU Library was introduced in [1]. This application note demonstrates the usage of the library on a simple application running on MPC5674F evaluation board. For evaluation purposes, the FreeMASTER GUI together with a logic analyzer enables run-time monitoring and application control.

5.1. References

[1] Engine Control eTPU Library, http://www.freesccale.com, AN4907.

[2] MPC567XEVBMB Evaluation Base Board Hardware User Guide, Axiom Manufacturing, http://www.axman.com.

[3] MPC567XACAT516 Demonstration Module Hardware User Guide, Axiom Manufacturing, http://www.axman.com.

[4] General C Functions for the eTPU, http://www.freesccale.com, AN2864.

[5] Pad Configuration and GPIO Driver for MPC5500, http://www.freesccale.com, AN2855.

[6] FreeMASTER Run-Time Debugging Tool, http://www.freescale.com/freemaster.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4908
Rev. 1
08/2015