life.augmented

AN5552

Application note

Generate PWM signals with GTM

Introduction

This document is intended to give information, method and guideline to generate simple and complex PWM signals using the GTM IP available in the SPC57x and SPC58x MCU. It provides all tools to create PWM signal with specific duration, wave and frequency.

The document gives all the information and method to generate Simple and Complex PWM signals using the tool SPC5Studio but also the needed code to integrate in a PWM project outside the tool.

Using SPC5Studio, the GUI approach is user friendly and very easy to use, few steps giving keys information to develop the project application.

A set of APIs are available to be used in the user code to customize the GUI configuration parameter and the application developing.
The online documentation is available to help the user in his project application development.

1 Scope

This document describes all steps to generate simple PWM signals using different GTM-IP submodule and same sub-module but with different approach.

The devices under analysis are listed in Table 1:

Table 1. Device list

Device	Part Number
SPC574Kx	SPC574K72E5, SPC574K72E7
	SPC574K70E5, SPC574K70E7
SPC572Lxx	SPC572L64F2, SPC572L64E3
	SPC572L60F2, SPC572L60E3
SPC58xEx	SPC58EE80E7, SPC58NE80E7, SPC58EE84E7, SPC58NE84E7
	SPC58EE80C3, SPC58NE80C3, SPC58EE84C3, SPC58NE84C3
	SPC58NE84H0
SPC58xNx	SPC584N80E7, SPC58EN80E7, SPC58EN84E7, SPC58NN84E7, SPC584N80C3, SPC58EN80C3, SPC58EN84C3, SPC58NN84C3

The GTM-IP version is listed in Table 2:

Table 2. GTM-IP list

Device	Part Number
SPC574Kx	GTM-IP 122
SPC572Lxx	GTM-IP 101
SPC58xEx	GTM-IP 343
SPC58xNx	GTM-IP 344

2 Overview

The PWM (Pulse Width Modulation) is a method to drive external actuators, power, electrical signal, etc. A PWM is a square wave, a signal switched between on and off. Simulating the portion of the time the signal spends “on” versus the time that the signal spends “off”.

The main elements to generate this kind of signal are:

- a clock used as source reference
- a period
- a duty

The duty is the duration (in the selected period) where the signal changes the polarity generating the output wave. Changing clock pre-scaler allows for a wide range of PWM durations with different resolution factors.

Figure 1. PWM

3 GTM & PWM: how to generate

PWM signal can be generated using different GTM submodules:

Table 3. PWM – GTM

PWM	GTM Sub-Module
Simple PWM generation	TOM
Complex PWM generation	FIFO, ATOM
Complex Output signal generation	MCS, ATOM

In all cases the GTM **CMU** (Clock Management Unit) sub-module must be configured in order to give the clock reference frequency.
Different CMU sub-module must be used:

Table 4. PWM – CMU

CMU sub-module	GTM – module
CMU_CLKx	ATOMx
CMU_FXCLKx	TOMx

3.1 GTM clock

The Clock Management Unit (CMU) is responsible for clock generation of the counters and of the GTM-IP.
The GTM clock is defined as a global input clock signal defined also as ***SYS_CLK***.
The sub block Global Clock Divider (*CMU_GCLK_EN*) can be used to divide the GTM-IP global input clock signal *SYS_CLK* into a common subdivided clock signal.
The CMU consists of three subunits that generate different clock sources for the whole GTM-IP:

- Configurable Clock Generation (CFGU)
- Fixed Clock Generation (FXU)
- External Clock Generation (EGU)

The GTM SYS_CLK clock is the MCU Clock: ***PER_CLK***

3.2 PWM formula

Table 5. PWM symbol

Symbol	Description
f_{out}	PWM output frequency
p	PWM period
CLK_SRC	Clock source frequency
T_{out}	Time

3.2.1 Output frequency

Starting from the CLK_SRC and from the PWM period p (in number of ticks), the PWM output frequency can be calculated as:

$$f_{out} = \frac{CLK\_SRC}{p} \quad (1)$$

Example:

CLK_SRC = 1 MHz

p = 10000 (ticks)

f_{out} = ?

$$f_{out} = \frac{1000000}{1000} = 100\ Hz$$

3.2.2 Period in second

From the PWM output frequency, the PWM period (in second) can be calculated as:

$$T_{out} = \frac{1}{f_{out}} \quad (2)$$

Example:

f_{out} = 1 KHz

T_{out} = ?

$$T_{out} = \frac{1}{1000} = 1\ ms$$

3.2.3 Period in ticks

Starting from the CLK_SRC and from the period (in second) or the output frequency, the period as number of ticks to set in the register can be calculated as:

- Starting from period

$$p = CLK\_SRC \times T_{out} \quad (3)$$

Example:

CLK_SRC = 1 MHz

T_{out} = 12 ms

p = ?

$$p = 1000000 \times 0.012 = 12000\ (ticks)$$

- Starting from frequency

$$p = \frac{CLK\_SRC}{f_{out}} \quad (4)$$

Example:

CLK_SRC = 1 MHz

f_{out} = 100 KHz

p = ?

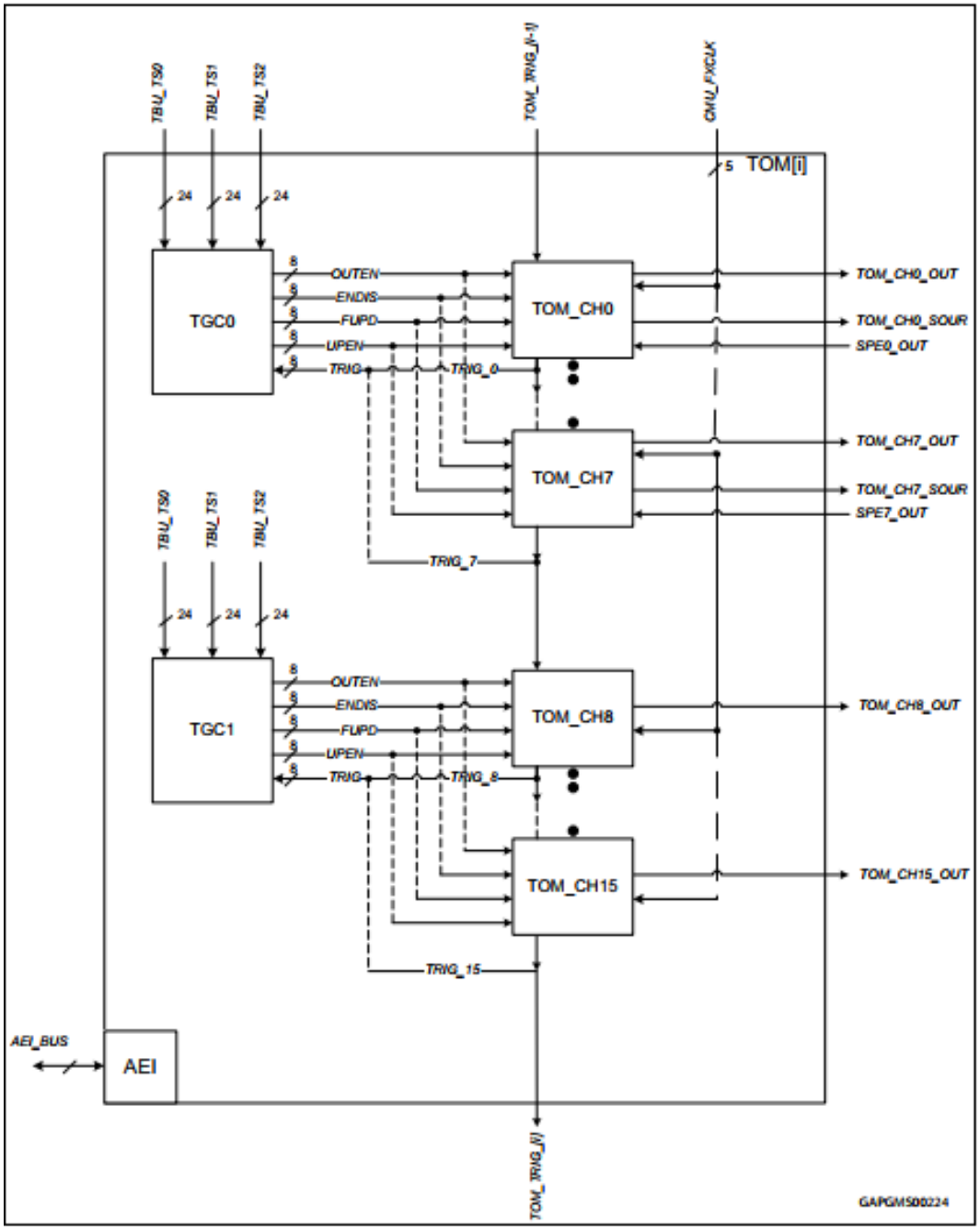
$$p = \frac{1000000}{100000} = 10$$

4 Simple PWM

A simple PWM can be generated using the TOM submodule (Timer Output Module).

The Timer Output Module (TOM) offers up to 16 independent channels (index x) to generate simple PWM signals at each output pin *TOM[i]_CH[x]_OUT*.

Figure 2. TOM Architecture



The two submodules **TGC0** and **TGC1** are global channel control units that control the enabling/disabling of the channels and their outputs as well as the update of their period and duty cycle register.

Each individual TOM channel includes a Counter Compare Unit 0 (CCU0), a Counter Compare Unit 1 (CCU1) and the Signal Output Generation Unit (SOU).

The CCU0 contains a counter **CN0** which is clocked with one of the selected input frequencies (*CMU_FXCLK*) provided from outside of the submodule.

When the counter register **CN0** is greater than or equal to the register **CM0,** the subunit CCU0 triggers the SOU subunit and the succeeding TOM submodule channel.

In the subunit CCU1 the counter register **CN0** is compared with the value of register **CM1**. If **CN0** is greater than or equal to **CM1** the subunit CCU1 triggers the SOU subunit.

Different values of CM0, CM1 and CN0 establish the characteristics of our PWM, e.g. duty 100% or duty 0% etc.

4.1 Configuration step

The source clock for the TOM sub-module is the CMU Fixed clock generation sub-unit (FXU) generating predefined non-configurable clocks *CMU_FXCLK[y]* (y: 0...4). The *CMU_FXCLK[y]* signals are derived from the *CMU_GCLK_EN* signal generated by the global clock divider. The dividing factors are defined as 2^0, 2^4, 2^8, 2^{12}, and 2^{16}.

Steps:

1. Configure the source FXU clock
2. Enable CMU unit
3. Configure specific TOM registers

Example:

Generate a PWM signal with output frequency of 600 Hz and a duty of 66 %.

MCU used: SPC574K with a PER_CLK of 80 MHz.

- The FXU Clock source selected: 80 MHz
- TOM source clock ***CMU_FXCLK[1]***: 5 MHz
- f_{out} = 600 Hz
- Period:

$$p = \frac{CLK\_SCR}{f_{out}} = \frac{5000000}{600} = 8334$$

- Duty = 66 % of p = 5500

4.1.1 PWM code

Figure 3. CMU settings

```
/*#define GTM_CMU(*(volatile struct GTM_CMU_tag *)0xFFD00300UL)*/

void CMU_clock_setup(void) {
      /* Disable all Clock - all internal clock counters will be reset */
      GTM_CMU.CLK_EN.R = 0x00000000UL;
      GTM_CMU.GCLK_NUM.R = 1UL;
      GTM_CMU.GCLK_NUM.R = 1UL;
      GTM_CMU.GCLK_DEN.R = 1UL;
      /*Configure the FXU (Fixed Clock  Generation) subunit */
      GTM_CMU.FXCLK_CTRL.R = 0UL; /*Global clock*/
      /*Enalbe cmu init with clock selected (FXU) */
      GTM_CMU.CLK_EN.R = 0x00800000; //(2 << 22)

}
```

Figure 4. **TOM settings**

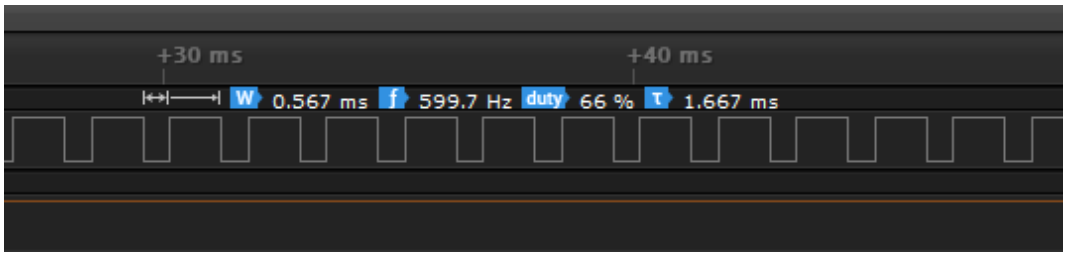
```
void TOM_config(void) {

        GTM_TOM_0.CH1_CTRL.R = 0x00001800;      /* CMU_FXCLK(1) and SL = High */
        GTM_TOM_0.CH1_SR0.R = 8334;             /* Period                     */
        GTM_TOM_0.CH1_SR1.R = 5500;             /* Duty                       */
        GTM_TOM_0.TGC0_GLB_CTRL.R = 0x00080001; /* ATOM0 CH1 and HOST_TRIG    */

        /*Enable Channel ATOM0 Channel 1 */
        GTM_TOM_0.TGC0_ENDIS_CTRL.R = 0x00000008;
        GTM_TOM_0.TGC0_ENDIS_STAT.R = 0x00000008;
        GTM_TOM_0.TGC0_OUTEN_CTRL.R = 0x00000008;
        GTM_TOM_0.TGC0_OUTEN_STAT.R = 0x00000008;
}
```

Result:

Figure 5. **Simple PWM**



4.1.2 Auxiliary code

Figure 6. **Initialization code**

```
static void MCU_Init(void) {
	uint8_t mode;
	/*SPC5_GTMINT_PCTL 128 */
	/*RUN mode 17 = SPC5_ME_PCTL_RUN(1) | SPC5_ME_PCTL_LP(2)*/
	MC_ME.PCTL[128].R = 17UL;
	mode = (uint8_t)MC_ME.MCTL.B.TARGET_MODE;
	/* Clearing status register bits */
	MC_ME.IS.R = 0x3FU;
	/* Starts a transition process.*/
	MC_ME.MCTL.R = ((((uint32_t)(mode)) << 28) | 0x5AF0UL);
	MC_ME.MCTL.R = ((((uint32_t)(mode)) << 28) | 0xA50FUL);
	/* Waits for the mode switch or an error condition.*/
	while (MC_ME.IS.R == 0U) {
		;
	}
	/* Check if no error during mode switch */
	if (MC_ME.IS.B.I_MTC != 1U) {
		irqSysHalt();
	}

}

static void GTM_Init(void) {
	GTMINT.MCR.R = 0x0UL;      /* Enable GTM Module, MDIS = 0 */
	GTM.CTRL.R = 0x0UL;        /* Clean GTM Control Register  */
	GTM.RST.R = 0x00000001UL; /* Reset All GTM devices       */
	if ((GTM.REV.R & 0xFFFFF000UL) != 0x12215000UL) {
		for(;;){}
	}
	GTMD.ip = 0x12215000UL;
	GTMD.gtm = &GTM;
	/*
	 * After global reset mechanism for triggering interrupts
	 * with IRQ_FORCINT, MCS RAM reset and MCS set scheduling
	 * option is globally disabled.
	 * Explicitly enabled it by clearing the bit RF_PROT.
	 */
	GTMD.gtm->CTRL.B.RF_PROT = 0UL;
	ICMD1.icm = &(GTM_ICM);

}
```

The main loop defines the PIN and calls/invokes the previous routines:

Figure 7. Main loop

```
#include “gtm.h”


/*PIN setup PF[0]= PAD[80]
   * [2..3]   OERC= 0 (Weak)
   * [5..7]   ODC = 2 (Push-pull)
   * [8]      SMC = 1
   * [12]     IBE = 1
   * [24..31] SSS = 8
   */
  SIUL2.MSCR_IO[80].R = 0x02880008;
  SIUL2.GPDO[80].R = 0x0;
  MCU_Init();
  GTM_Init();
  CMU_clock_setup();

  TOM_config();
```

Figure 8. **Header file gtm.h**

```
#include <platform.h>
#include "SPC574K_GTM.h"
#include <irq.h>
#define GTM_TAG                    struct GTM_tag
#define GTM_CMU_TAG                struct GTM_CMU_tag
#define GTM_ICM_TAG                struct GTM_ICM_tag
#define GTM_TOM_TAG                struct GTM_TOM_tag
#define SPC5_GTM_ICM_TOM0          25UL
#define SPC5_GTM_ICM_TOM0_CHANNEL_1    0x00000002UL
typedef struct {
  volatile GTM_ICM_TAG *icm;
   void *priv;
}GTM_ICMDriver;
extern GTM_ICMDriver ICMD1;
#define SPC5_GTM_TOM_INT_PRIORITY        15
#define SPC5_TOM0_EVENT0_HANDLER         vector813
#define SPC5_TOM0_EVENT0_INT_NUMBER      813
#define SPC5_TOM0_EVENT0_INT_PRIORITY    SPC5_GTM_TOM_INT_PRIORITY
IRQ_HANDLER(SPC5_TOM0_EVENT0_HANDLER);
typedef struct {
        uint32_t aru;
        uint32_t atom;
        uint32_t dpll;
        uint32_t tom;
        uint32_t tim;
        uint32_t psm;
        uint32_t brc;
        uint32_t mcs;
}GTM_INT_PR;
typedef struct {
        volatile GTM_TAG *gtm;
        uint32_t ip;
        uint32_t gtm_auxin[3];
        const GTM_INT_PR *gtm_int_pr;
}GTMDriver;
extern GTMDriver GTMD;
#define GTM_ERROR    0x1U
```

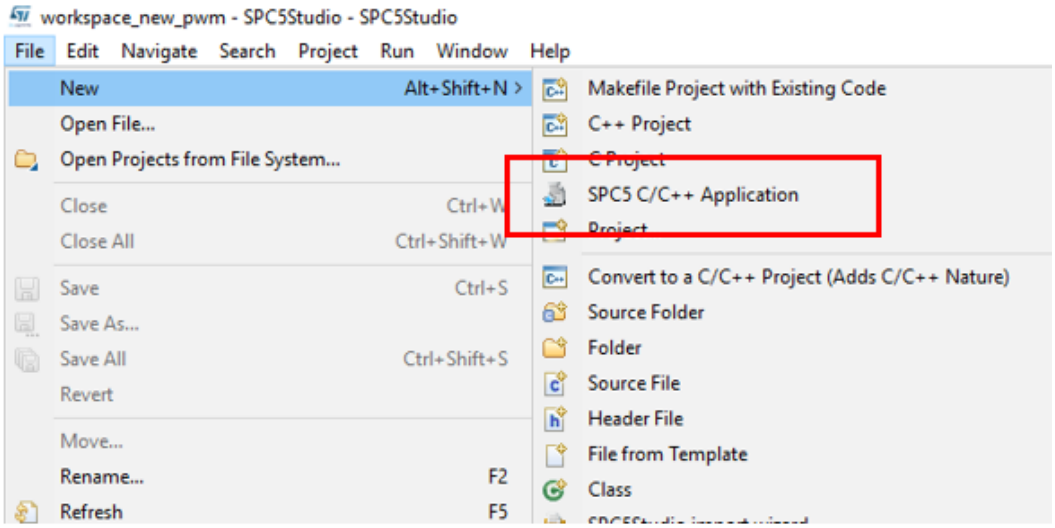
4.2 SPC5Studio

Configuration and execution are more user-friendly using SPC5Studio.
Few steps and the PWM are ready to be executed in your selected platform.

4.2.1 Create PWM project

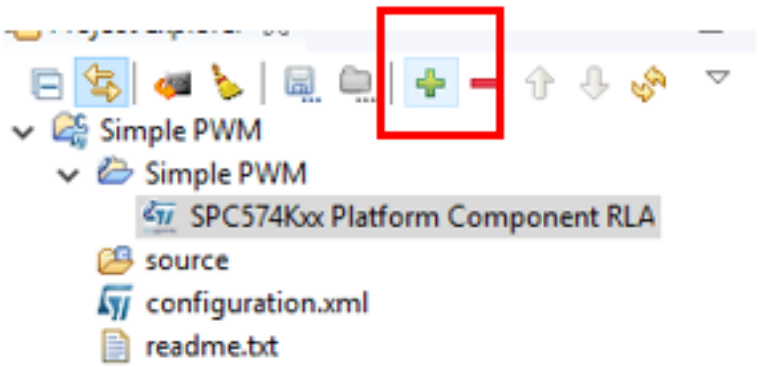
Create a new SPC5 project:

Figure 9. SPC5Studio new project



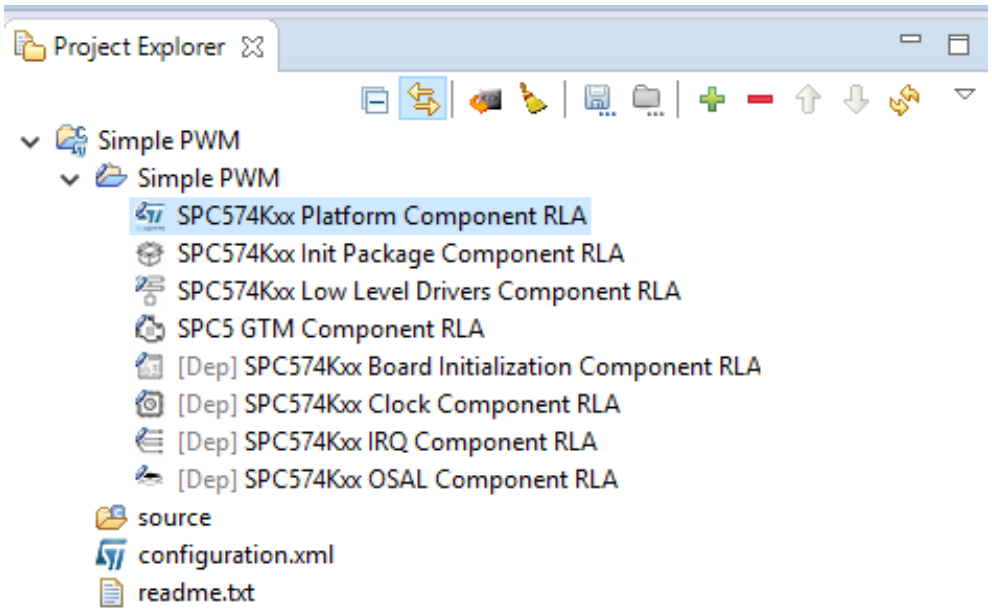
Choose your platform and add the components preferred (the Component Init is mandatory):

Figure 10. Component Add



Your project ready.

Figure 11. PWM project ready



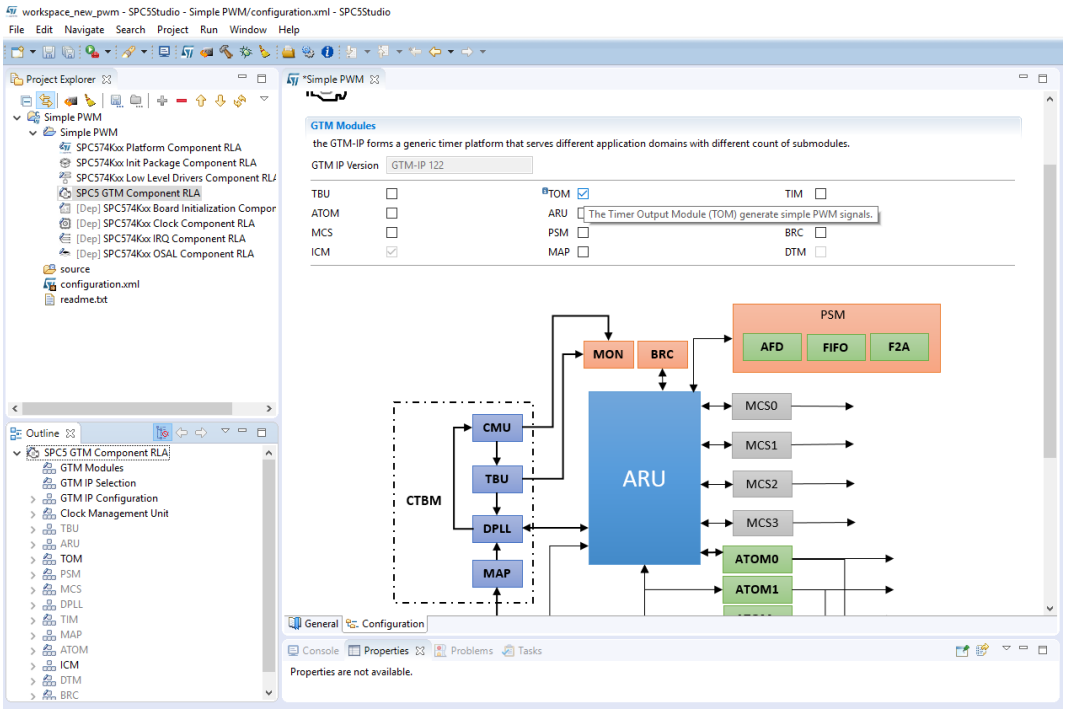
4.2.2 Configure SPC5Studio project

In order to generate a PWM signal, the following steps are needed

1. Enable GTM Module

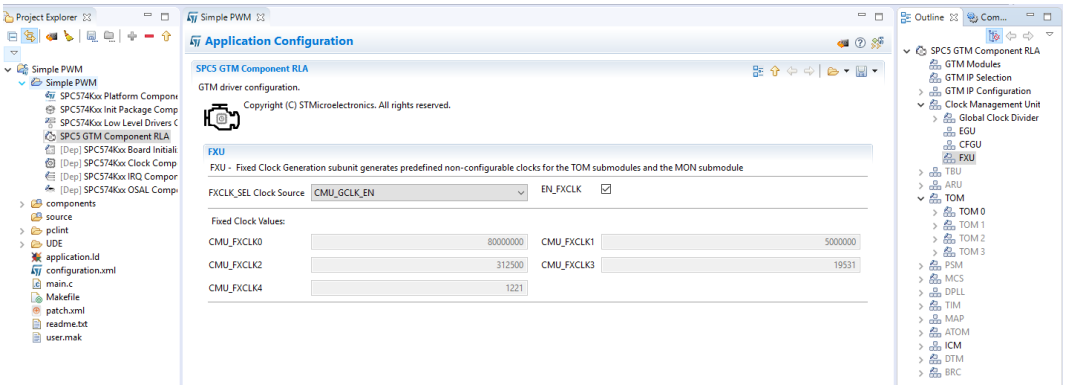
   Enable TOM module in the GTM component:

Figure 12. GTM modules



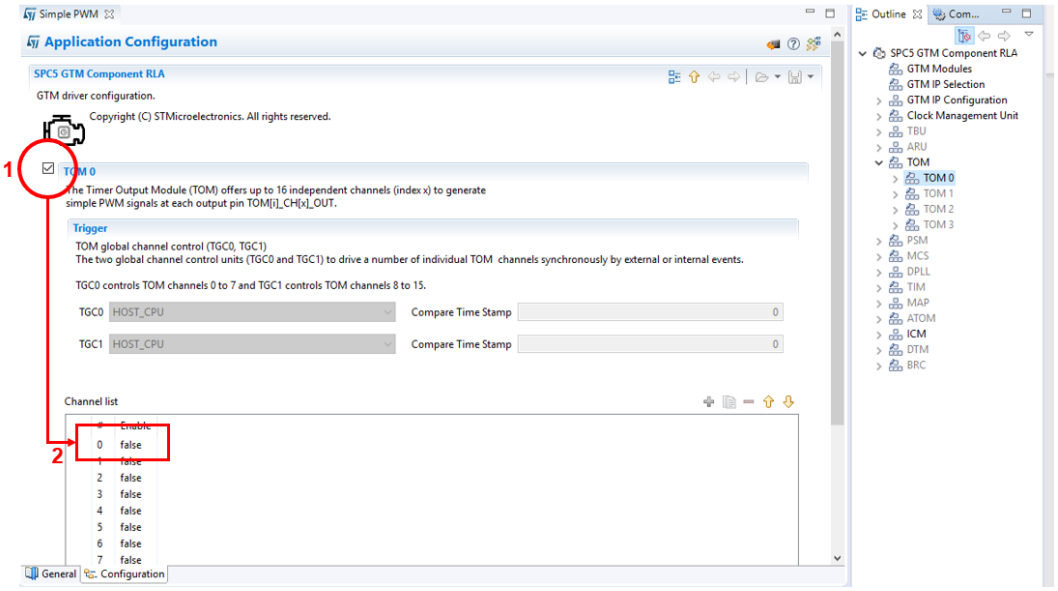
2. Configure CMU unit
   TOM use the FXU (Fixed Clock Generation) sub-unit:

Figure 13. FXU Configuration



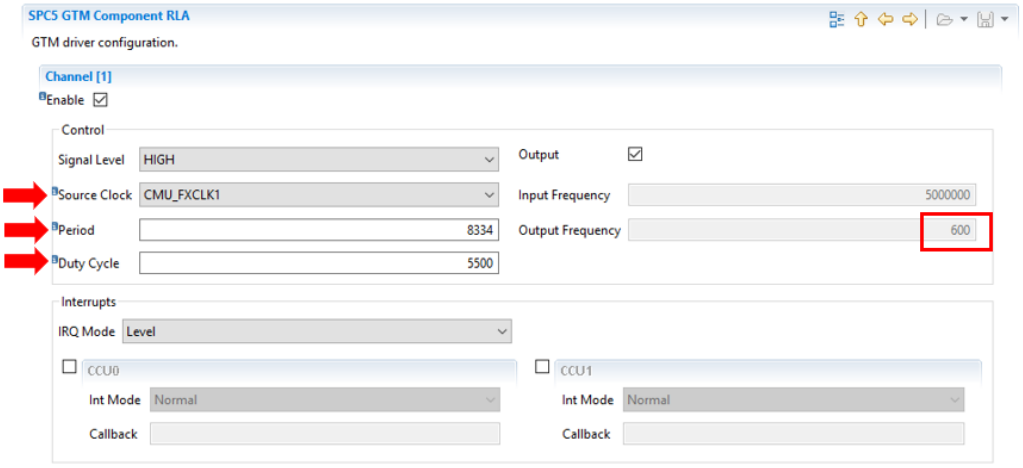
3. TOM configuration
   Enable TOM0 IP and click on Channel List, Channel 0 to Enable/ Configure it:

Figure 14. TOM Configuration



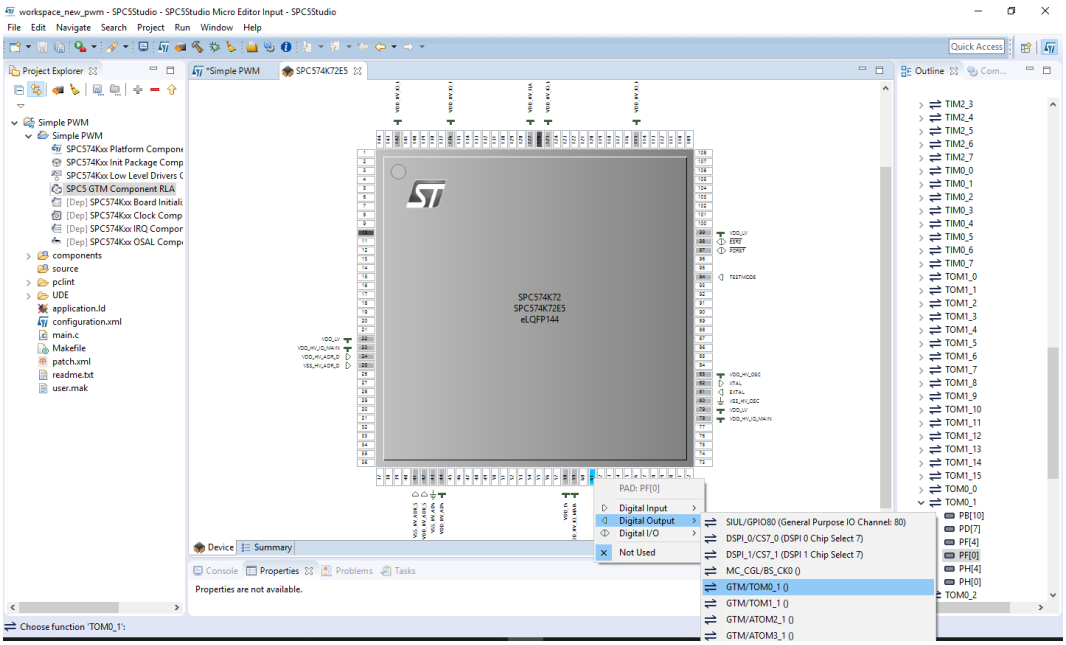
4. Enable and configure TOM0 Channel 1
   Configure the channel in order to have a PWM with an output frequency of 600 Hz and duty of 66 %:

Figure 15. TOM channel settings



5. Enable PIN PF[0] with TOM0_1 functionality
   Configuration done using the Pinmap wizard:

Figure 16. Pin setting



6. Populate your main application
   Enable TOM module in the GTM component

Figure 17. **Main.c**

```
int main(void) {
	componentsInit();
	/* Enable Interrupts */
	irqIsrEnable();

	/*Start CMU sub module*/
	gtm_cmuStart(&CMUD1);
	/*Start TOM channel configured*/
	gtm_tomStart();

	/* Application main loop.*/
	for ( ; ; ) {
	}
}
```

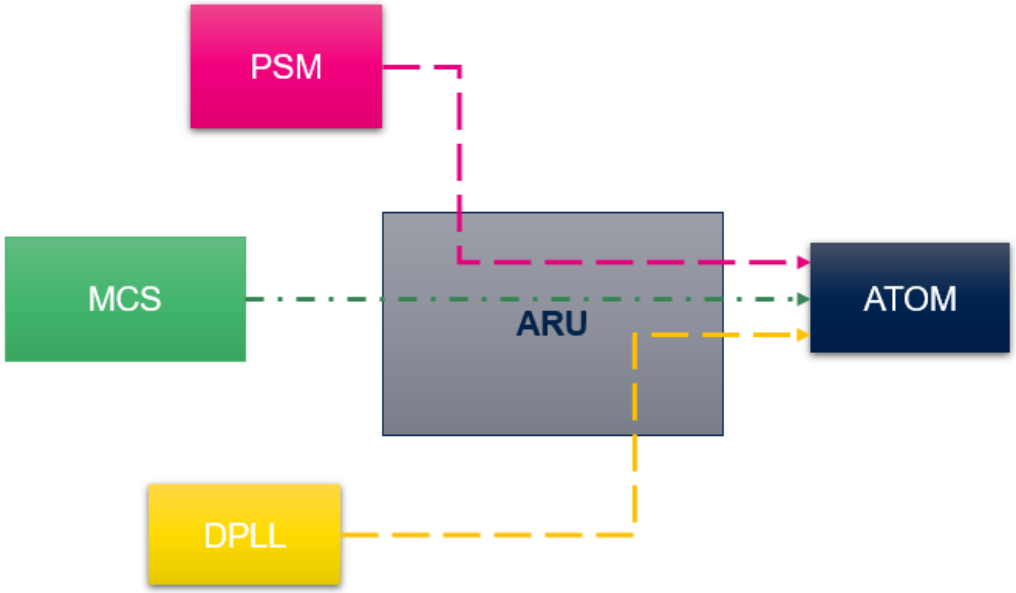
5 Complex PWM

Complex PWM signal can be generated using the GTM ATOM (ARU-connected Timer Output Module) unit.

5.1 GTM ATOM

The ARU-connected Timer Output Module (ATOM) can generate complex output signals without CPU interaction thanks to its connectivity to the ARU. Typically, output signal characteristics are provided over the ARU connection through submodules connected to ARU like e.g. the MCS, DPLL or PSM.

Figure 18. ATOM and GTM interaction



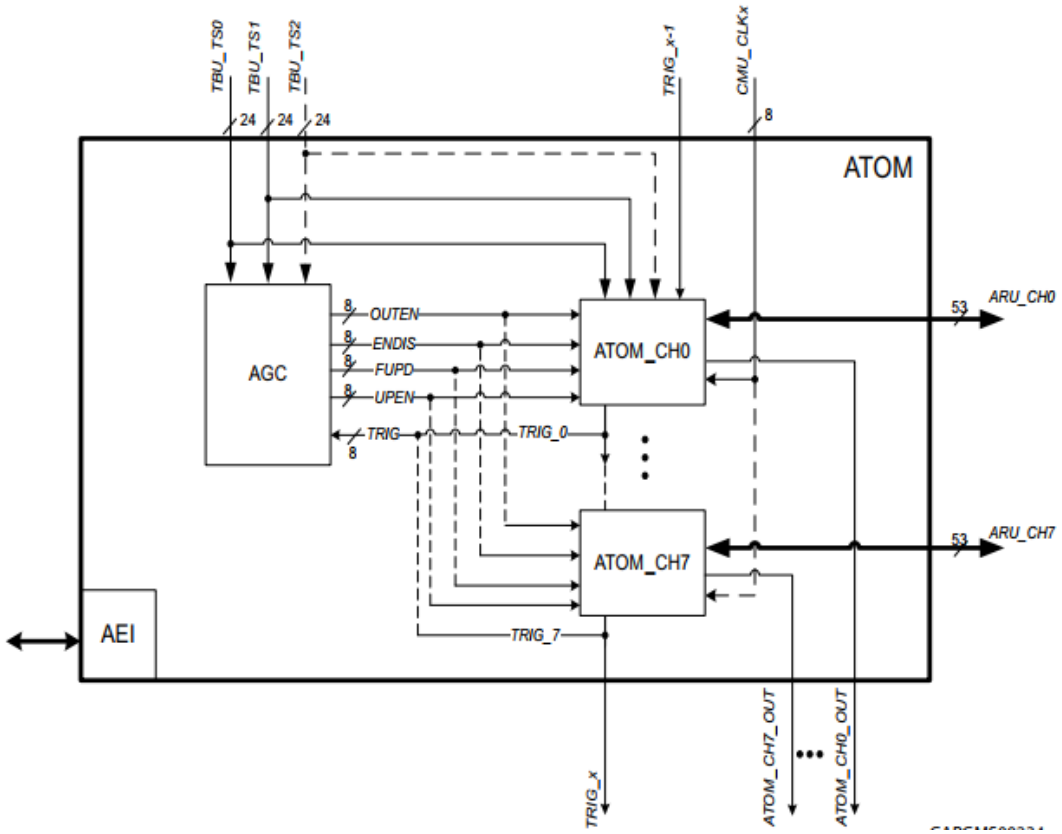
The architecture of the ATOM submodule is like the TOM submodule, but there are some differences. First, the ATOM integrates only eight output channels. Hence, there exists one ATOM Global Control subunit (AGC) for the ATOM channels. The ATOM is connected to the ARU and can set up individual read requests from the ARU and write requests to the ARU.

Each ATOM channel provides four modes of operation:

- ATOM Signal Output Mode Immediate (SOMI)
- ATOM Signal Output Mode Compare (SOMC)
- ATOM Signal Output Mode PWM (SOMP)
- ATOM Signal Output Mode Serial (SOMS)

In our case only the **SOMP** mode will be analyzed. A block diagram of the ATOM submodule is depicted in the Figure 19:

Figure 19. **ATOM block diagram**



The input clocks for the ATOM channels come from the configurable *CMU_CLKx* signals of the CMU submodule. This allows to select a programmable input clock for the ATOM channel counters.

Each ATOM channel provides the *operation* and *shadow* register sets. With this architecture it is possible to work with the operation register set, while the shadow register set can be reloaded with new parameters over CPU and/or ARU.

When update via ARU is selected, it is possible to configure if both shadow registers are updated via ARU or only one of the shadow registers is updated for SOMP mode.

5.2 ATOM SOMP mode

In ATOM Signal Output Mode PWM (SOMP) the ATOM submodule channel can generate complex PWM signals with different duty cycles and periods. Duty cycles and periods can be changed synchronously and asynchronously. Synchronous change of the duty cycle and/or period means that the duty cycle or period duration changes after the end of the preceding period. An asynchronous change of period and/or duty cycle means that the duration changes during the actual running PWM period.

The register involved in the PWM generation are:

- **CN0**: counter register
- **CM0**: holds the duration of the period in clock ticks of the selected CMU clock
- **CM1**: holds the duration of the duty cycle in clock ticks of the selected CMU clock

5.2.1 SOMP one-shot mode

The ATOM channel can operate in One-shot mode when the **OSM** bit is set in the channel control register. One-shot mode means that a single pulse with the pulse level defined in bit SL is generated on the output line.

First the channel has to be enabled by setting the corresponding **ENDIS_STAT** value. In One-shot mode the counter **CN0** will not be incremented once the channel is enabled. A write access to the register **CN0** triggers the start of pulse generation (i.e. the increment of the counter register **CN0**).

5.2.2 GTM code

The code is an extension of the previous one, the TOM code could be replaced with the ATOM.
Generate a Complex PWM using ATOM0 Channel 1 with output frequency of 100 Hz and a Duty of 80 %
The clock source is CLK_SRC0 using a clock divider set to 80, output frequency is: 1 MHz.
The PWM Period will 10000 ticks and Duty = 8000.

$$p = \frac{CLK\_SRC}{f_{out}} = \frac{1000000}{100} = 10000$$

Define a new routine with the ATOM configuration:

Figure 20. ATOM setting

```
void ATOM_config(void) {
      /*Select the CFCU CLK_SRC0 */
    GTM_CMU.CLK_0_CTRL.R = 80UL - 1UL; /* Selected Clock divider - 1 */
    GTM_CMU.CLK_EN.R |= 2UL;           /* Enable CLK_SRC0            */

    /* Set Period and Duty */
    GTM_ATOM_0.CH1_CTRL.R = 0x00000802;        /*SL=1, MODE= 2(SOMP)  */
    GTM_ATOM_0.CH1_CM0.R =  10000;
    GTM_ATOM_0.CH1_CM1.R =  8000;
    GTM_ATOM_0.CH1_SR0.R =  10000;
    GTM_ATOM_0.CH1_SR1.R =  8000;
    GTM_ATOM_0.AGC_GLB_CTRL.R =   0x00080000;  /* Enable UPEN_CTRL1   */
    GTM_ATOM_0.AGC_ENDIS_STAT.R = 0x00000008;  /* Enable ATOM0_ CH1   */
    GTM_ATOM_0.AGC_OUTEN_STAT.R = 0x00000008;  /* Enable ATOM0_ CH1   */

}
```

Main define the PIN PD[15] and the ATOM_config() routine:

Figure 21. ATOM main loop

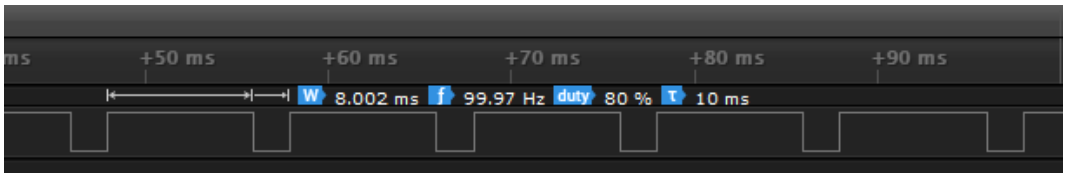
```
…
/*PD15 ATOM0_1 */
  SIUL2.MSCR_IO[63].R = 0x0288000A; /*SSS = 0xA*/
  SIUL2.GPDO[63].R = 0x0;

  MCU_Init();
  GTM_Init();
  CMU_clock_setup();
  TOM_config();
  ATOM_config();

....
```

Result:

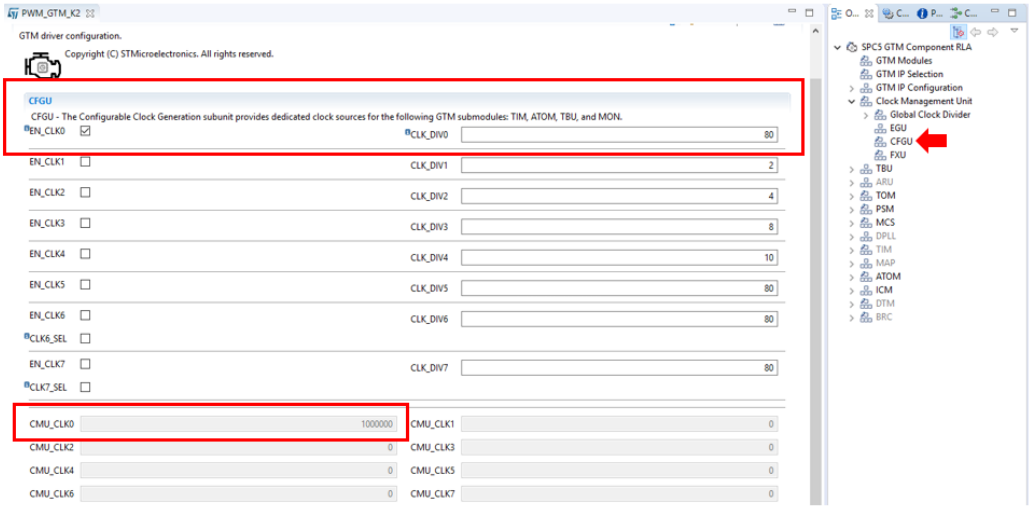
Figure 22. Complex PWM – ATOM



5.3 SPC5STUDIO and ATOM SOMP

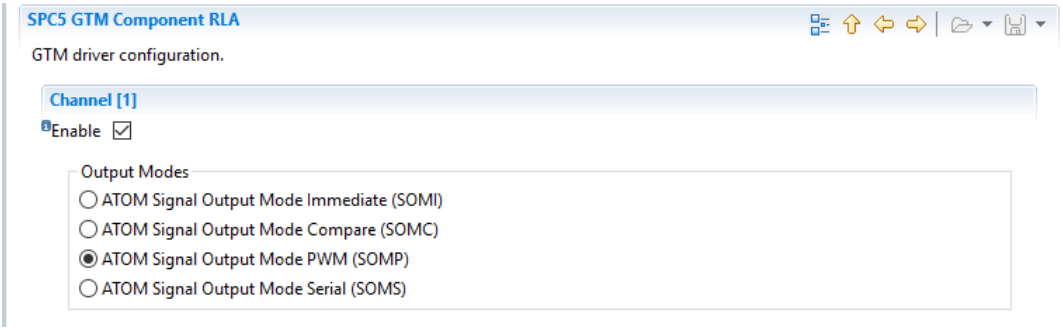
The clock reference for the ATOM is the CMU sub-unit **CFGU**.
Enable the subunit in the CMU GUI:

Figure 23. CFGU configuration



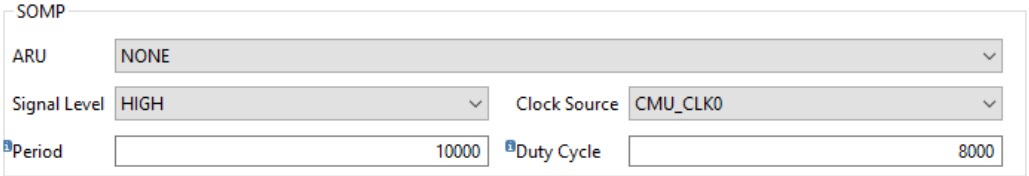
Configure the ATOM Channel, enabling the Channel 1:

Figure 24. ATOM configuration



The last step is to add the function: ***gtm_atomStart()*** in your main loop and define the pin using the Pinmap Wizard.
Result:

Figure 25. Complex PWM



5.4 ATOM and PSM

Complex PWM can be generated using multiple GTM sub-units. The connection between the different sub-units is performed through the ARU sub-unit.

5.4.1 ARU (Advanced Routing Unit)

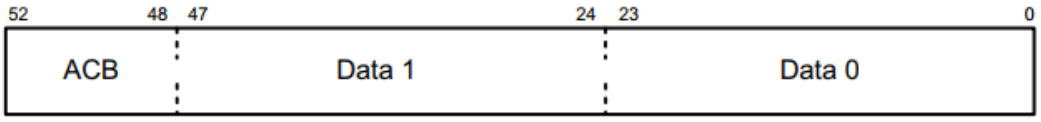
A key point of the GTM-IP is the routing mechanism of the ARU submodule for data streams. Each data word transferred between the ARU and its connected submodule is 53-bit wide.

Each module that is connected to the ARU may provide an arbitrary number of ARU write channels and an arbitrary number of ARU read channels. In the following, the ARU write channels are named ***data sources*** and the ARU read channels are named ***data destinations***.

The concept of the ARU intends to provide a flexible and resource efficient way for connecting any data source to an arbitrary data destination. In order to save resource costs, the ARU implements a data router with serialized connectivity providing the same interconnection flexibility.

A connection between a data source and a data destination is also called a ***data stream***.

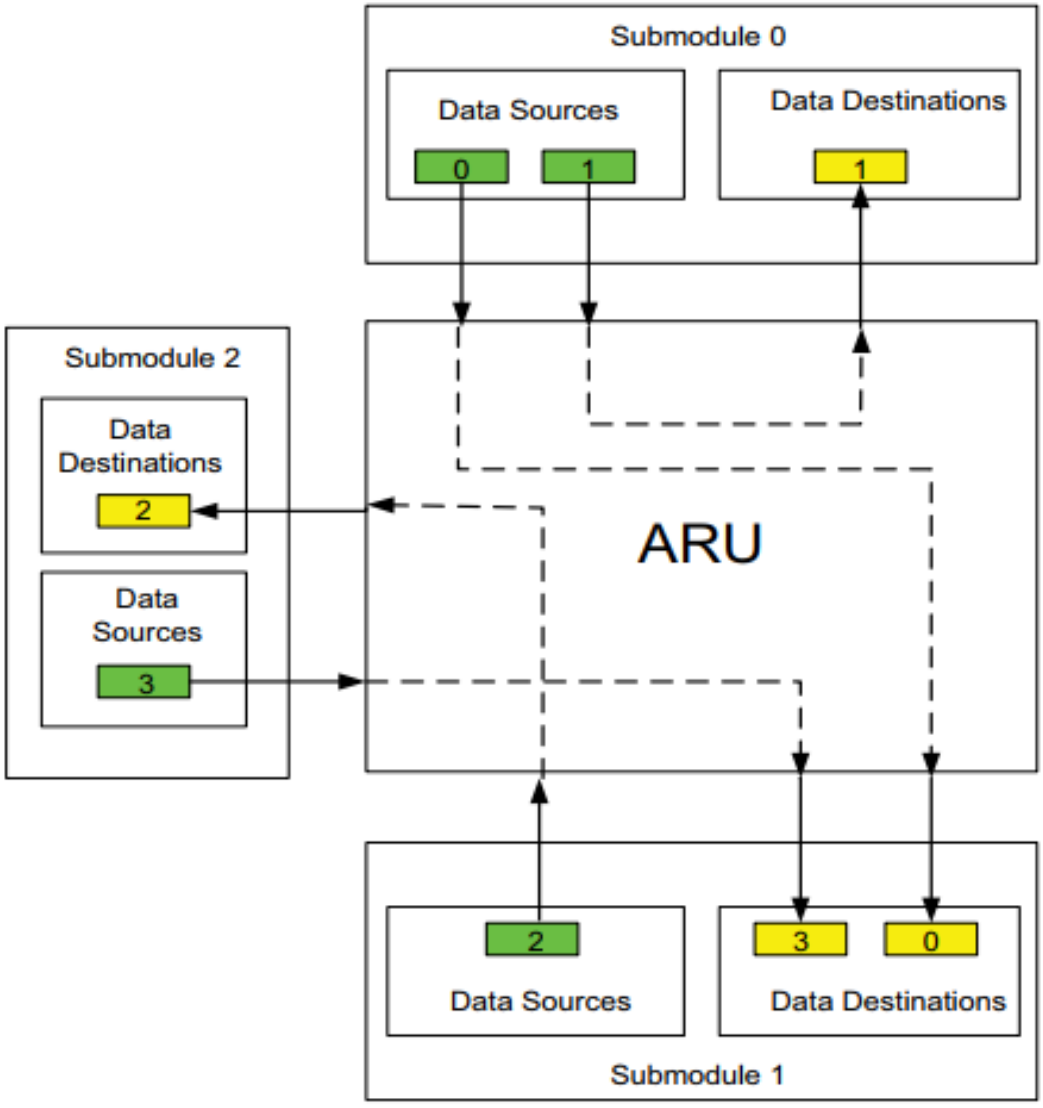
Figure 26. ARU data stream



The configuration of the data streams is realized according to the following manner:

- Each data source has its fixed and unique source address. The fixed address of each data source is pointed out by the numbers in the *green* boxes.
- The connection from a specific data source to a specific data destination is defined by configuring the corresponding address of a data source in the desired data destination. The configured address of each data destination is pointed out by the numbers in the *yellow* boxes.

Figure 27. **ARU block diagram**



5.4.2 PSM (Parameter Storage Module)

The PSM submodule consists of three subunits:

- AEI-to-FIFO Data Interface (AFD)

  The AFD submodule implements a data interface between the AEI bus and the FIFO submodule, which consists of eight logical FIFO channels.
- FIFO-to-ARU Interface (F2A)

  The F2A is the interface between the ARU and the FIFO submodule. Since the data width of the ARU (ARU word) is 53-bit (two 24-bit values and five control bits)
- FIFO

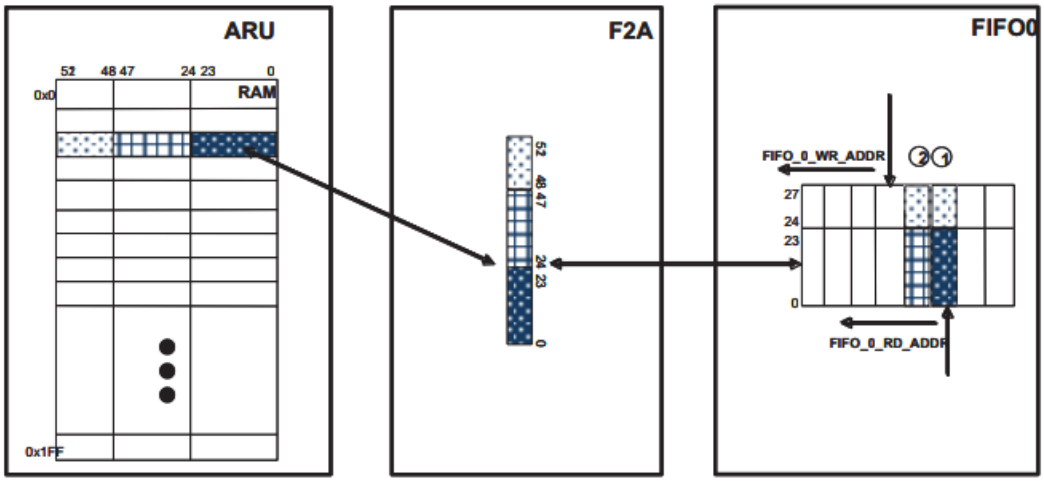
  Each logical FIFO represents a data stream between the submodules of the GTM and the microcontroller connected to AFD submodule

The PSM can be used as a data storage for incoming data characteristics or as parameter storage for outgoing data. This data is stored in a RAM that is logically located inside the FIFO subunit, but physically the RAM is implemented and integrated by the silicon vendor with his RAM implementation technology. Therefore, the GTM-IP provides the interface to the RAM at its module boundary. The AFD subunit is the interface between the FIFO and the GTM SoC system bus interface AEI. The F2A subunit is the interface between the FIFO subunit and the ARU.

Figure 28. PSM block

In the following figure the schema that describe how the data are transferred between ARU and FIFO:

Figure 29. Data transfer ARU - FIFO



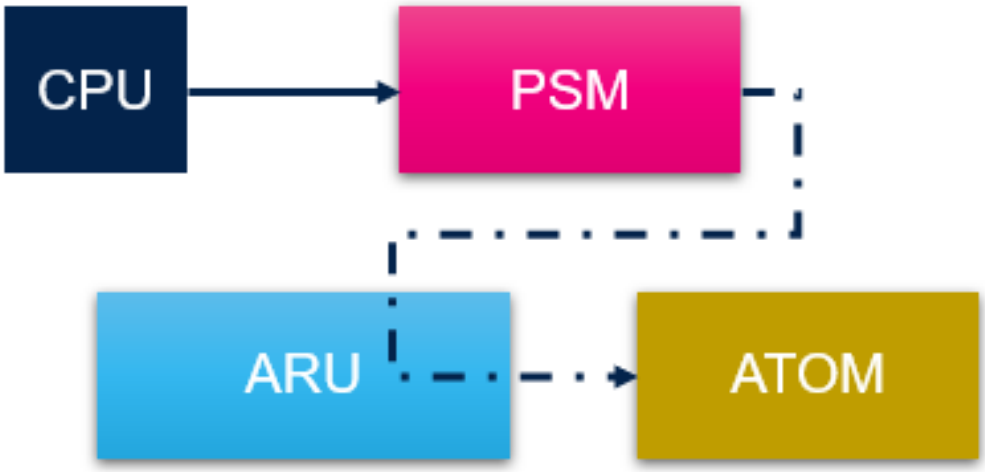
5.4.3 Complex PWM data block

The PWM specific data: Clock, Period and Duty must be uploaded in the PSM, with the appropriate sequence to create the ARU stream. The input data for the PWM/ATOM will be the ARU.

Data:

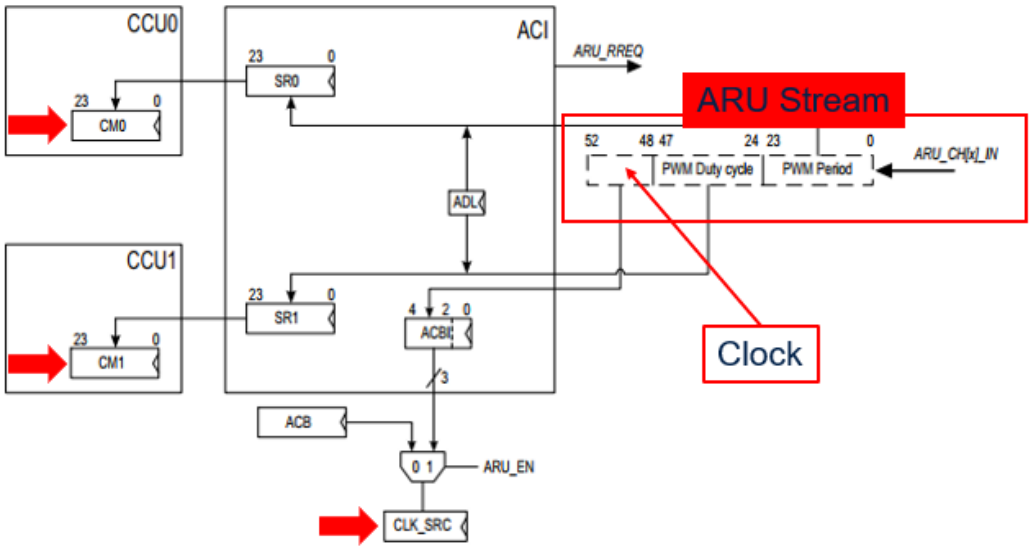
- Clock
- Period
- Duty

Figure 30. Complex PWM data block



How the ARU stream must be formatted in order generate PWM signal:

Figure 31. ARU data input stream for PWM



5.4.4 PSM code

Our goal is to generate a PWM signal using data from CPU stored in the PSM Channel0. These data will be Input for ATOM0 Channel2 thought ARU.

Clock: CLK_SRC0 (1Mhz), Period = 10000 tick, Duty 20 %.

In the ACB bit (bit [52…48] of ARU Stream) only the bit [4..2] are useful, in this specific case all ACB bit will “0” because the source clock is CLK0, in case of CLK5 then ACB bit will be:0x14 (b101 - 00).

PIN defined: PE[10]:PAD[74], ATOM0_2, SSS = 10

Figure 32. PSM code

```
/*PSM Initialization and Enable */
static void PSM_config(void) {
       GTM_FIFO_0.CHANNEL[0].CTRL.B.WULOCK = 1;    /* enables direct memory access by CPU  */
       GTM_FIFO_0.CHANNEL[0].CTRL.B.RBM    = 1;    /* FIFO works in Ring Buffer Mode       */
       GTM_FIFO_0.CHANNEL[0].CTRL.B.RAP    = 1;    /* RAM access priority                  */

       GTM_F2A_0.CH_STR_CFG[0].B.TMODE     = 2;    /* ARU both words in the PSM stream     */
       GTM_F2A_0.CH_STR_CFG[0].B.DIR       = 1;    /* FIFO to ARU direction                */

       /*Flush the PSM FIFO */
       GTM_FIFO_0.CHANNEL[0].CTRL.B.FLUSH = 1;
       while(GTM_FIFO_0.CHANNEL[0].CTRL.B.FLUSH != 0U){};

       GTM_FIFO_0.CHANNEL[0].START_ADDR.R = 0x0; /* Start Address */
       GTM_FIFO_0.CHANNEL[0].END_ADDR.R   = 0x2; /* End address   */

       /* Write PWM data Period in the BUFF_ACC (AFD interface)   */
       GTM_AFD_0.CH[0].BUF_ACC.R = 10000;                  /* PWM Period */
       GTM_AFD_0.CH[0].BUF_ACC.R = ((0x0 << 24) + 2000);   /* Duty and clock source in word b
[28..24] */

       GTM_F2A_0.ENABLE.R = 0x2;                           /* Enable PSM */

}
```

Extend ATOM routine with the ATOM0_2 settings.

ATOM source is the ARU FIFO address: 0x51

Figure 33. ATOM routine (CH2 settings)

```
/* ATOM0 CH2 Settings PWM-PSM */
    GTM_ATOM_0.CH2_CTRL.R = 0x0000080A;        /*SL=1, ARU_EN=1, MODE= 2(SOMP) */
    GTM_ATOM_0.CH2_CM0.R = 10000;
    GTM_ATOM_0.CH2_CM1.R = 2000;
    GTM_ATOM_0.CH2_SR0.R = 10000;
    GTM_ATOM_0.CH2_SR1.R = 2000;

    GTM_ATOM_0.CH2_RDADDR.R      = 0x01FE0051;    /* 0x51 FIF0 ARU address */
    GTM_ATOM_0.AGC_GLB_CTRL.R   |= 0x00200000;    /* Enable UPEN_CTRL2     */
    GTM_ATOM_0.AGC_ENDIS_STAT.R |= 0x00000020;    /* Enable ATOM0_ CH2     */

    GTM_ATOM_0.AGC_OUTEN_STAT.R |= 0x00000020;    /* Enable ATOM0_ CH2     */
```

Pin definition and main application:

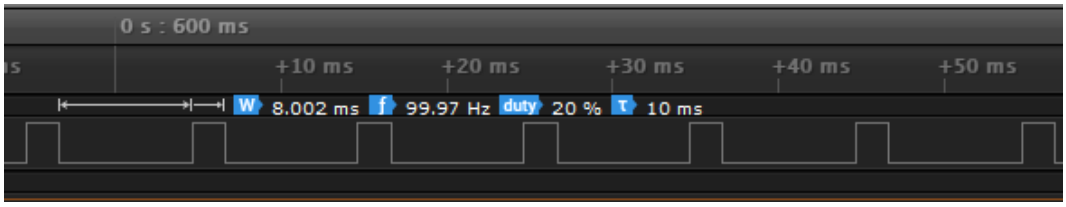
Figure 34. **Main loop ATOM-PSM**

```
…..
/*PE[10] ATOM0_2 */
  SIUL2.MSCR_IO[74].R = 0x0288000A; /*SSS = 0xA*/
  SIUL2.GPDO[74].R = 0x0;


  MCU_Init();
  GTM_Init();
  CMU_clock_setup();
  TOM_config();
  PSM_config();
  ATOM_config();
…..
```

Result:

Figure 35. **Complex PWM**



5.5 Complex PWM with SPC5Studio

In order to create a complex PWM signal, PSM and ATOM will be used.

5.5.1 Enable PSM interface

Figure 36. **PSM**

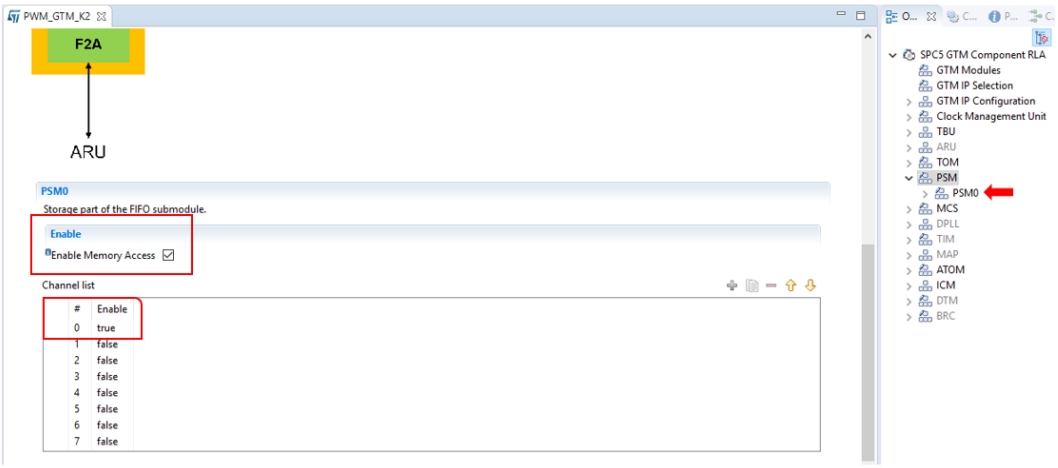


Figure 37. **PSM Channel0 settings**

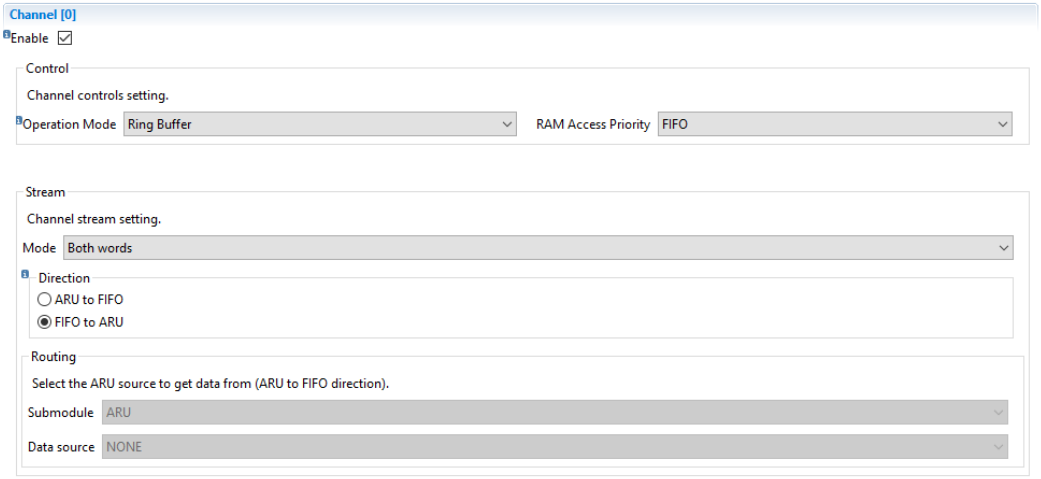
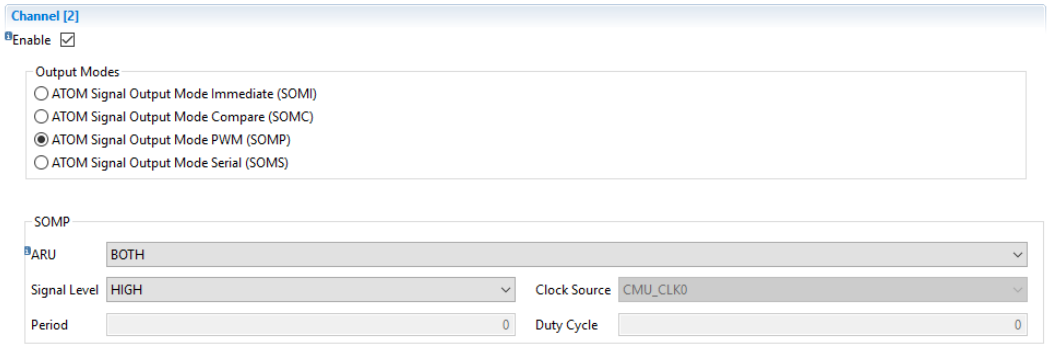


Figure 38. **ATOM0_2 settings**



Define PIN (e.g. PE[10]) using the Pinmap Wizard and add the GTM routines in the main loop:

Figure 39. Complex PWM - main loop

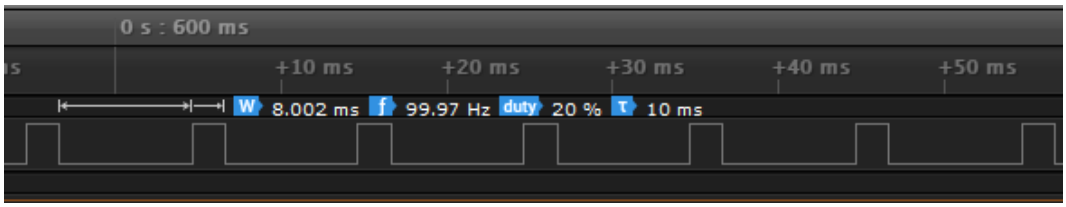
```
/******* ATOM0 Channel2 PWM with PSM data via ARU **********/

clk_src = ((uint32_t)SPC5_GTM_CMU_CLK0 << 2UL);
gtm_psmFlush(&PSMD1, PSM_CHANNEL0);
gtm_psmSetStartAddress(&PSMD1, PSM_CHANNEL0, 0);
gtm_psmSetEndAddress(&PSMD1, PSM_CHANNEL0, 2);
/* Period */
gtm_psmWrite(&PSMD1, PSM_CHANNEL0, 10000);
/* Duty cycle */
gtm_psmWrite(&PSMD1, PSM_CHANNEL0, 2000 + (clk_src << 24U));
/* Start providing data */
gtm_psmStart(&PSMD1, PSM_CHANNEL0);
gtm_atomSetDataSource(&ATOMD1, ATOM_CHANNEL2, SPC5_GTM_WRITE_ADDRESS_F2A0_F2A0_0);
gtm_atomStart(&ATOMD1, ATOM_CHANNEL2);

/*****************************************************************/
```

Result:

Figure 40. Complex SPC5Studio PWM



Appendix A Reference documents

Table 6. **Reference documents**

Doc Name	ID	Title
RM0361	025070	Generic Timer Module specification revision 1.5.5.1

Revision history

Table 7. **Document revision history**

Date	Version	Changes
28-Sep-2020	1	Initial release.

Contents

List of tables

List of figures