life.augmented

AN5693

Application note

GTM and MCS complex PWM signal

Introduction

This document is intended to give information, method and guideline to use the MCS GTM submodule and its integration with other GTM modules generating a complex PWM signals with specific duration, wave and frequency.

1 Scope

This document describes all steps to configure and use the MCS GTM submodule, a complex PWM signals will be generated as example.

The devices under analysis are listed in Table 1.

Table 1. Device list

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

The GTM-IP version are listed in Table 2.

Table 2. GTM-IP list

Device	Part Number	Specification Revision
SPC574Kx	GTM-IP 122	1.5.5.1
SPC572Lxx	GTM-IP 101	1.5.5.1
SPC58xEx	GTM-IP 343	3.1.5.1
SPC58xNx	GTM-IP 344	3.1.5.1

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 element 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 of clock pre-scaler allows for a wide range of PWM durations with different resolution factors.

Figure 1. PWM

3 GTM and MCS description

The Multi Channel Sequencer (MCS) submodule is a generic data processing module that is connected to the ARU.

One of its major applications is to calculate complex output sequences that may depend on or used in combination with other GTM submodules like ATOM.

Some applications may process data provided by the CPU within the MCS sub module, and the calculated results are sent to the outputs using the ATOM sub modules.

The MCS sub module mainly are connected to two RAM pages located outside of the MCS sub module.

The data path of the MCS is shared by eight so called MCS-channels, whereas each MCS channel executes a dedicated micro-program that is stored inside the RAM pages connected to the MCS sub module. The execution of the different MCS-channels is controlled by a central task scheduler.

Both RAM pages may contain arbitrary sized code and data sections that are accessible by all MCS-channels and the CPU via AEI.

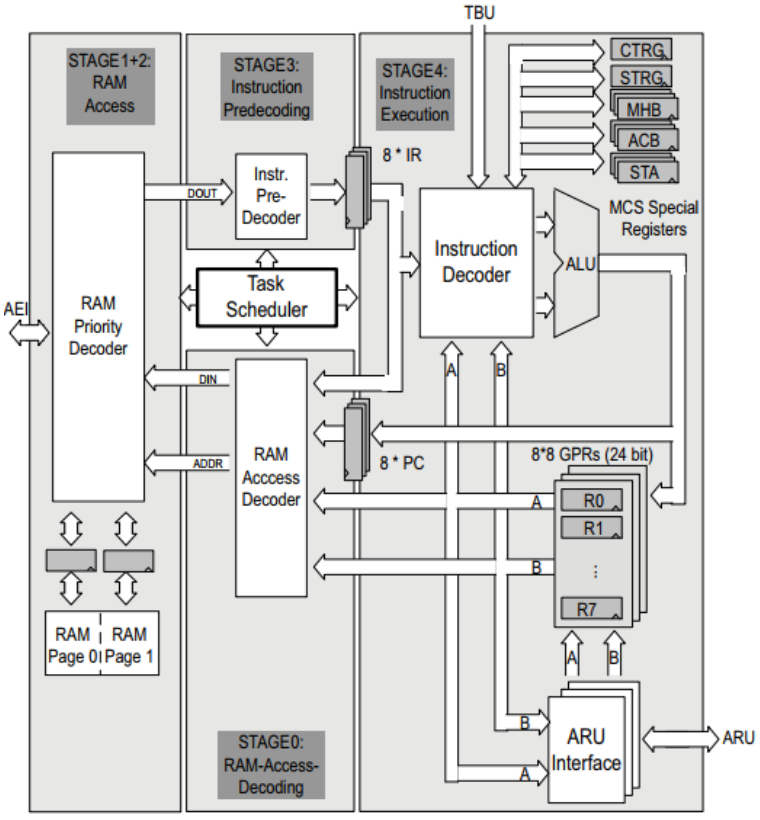
The address space of the MCS is divided into two seamless memory pages.

Memory page 0 begins from address 0 and ranges to address MP0-4 and memory page 1 ranges from MP0 to MP1-4. The parameters MP0 and MP1 are defined externally by the memory configuration sub module MCFG.

The MCS is a programmable RISC like sequencer module, which executes several parallel working tasks on a single processing core. An MCS-channel can also be considered as an individual task of a processor that is scheduled at a specific point in time.

Moreover, each MCS-channel has a dedicated ARU interface for communication with other ARU connected modules, an Instruction Register (IR), a Program Counter Register (PC), a Status Register (STA), an ARU Control Bit Register (ACB), a Memory High Byte Register (MHB) and a Register Bank with eight 24-bit general purpose registers (R0, R1...R7).

Figure 2. MCS architecture

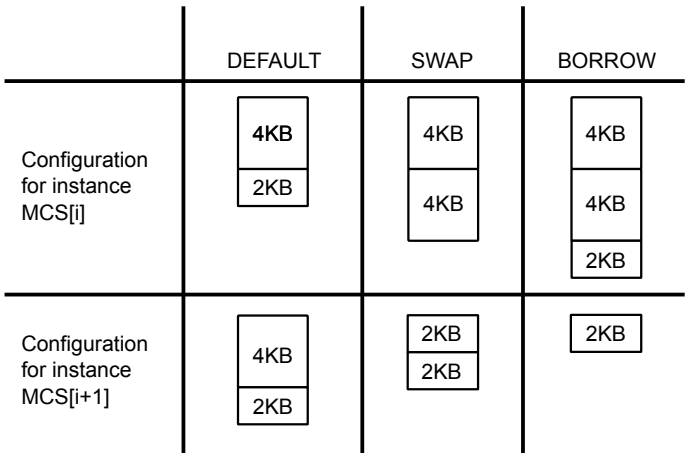


3.1 Memory configuration (MCFG)

The Memory Configuration submodule (MCFG) is an infrastructure module that organizes physical memory blocks and maps them to the instances of Multi Channel Sequencer (MCS) submodules. The default configuration maps a memory of size 1 K*32 bit = 4 KB to MCS memory page 0 and a memory of size 0.5 K*32 bit = 2 KB to MCS memory page 1.

In order to support different memory sizes for different MCS instances, the MCFG module provides two additional layout configurations for reorganization of memory pages between neighboring MCS modules.

Figure 3. MCS memory layout



The layout configuration SWAP is swapping the 2 KB memory page of the current MCS instance with the 4KB memory page of the successive MCS instance. Thus, the memory of the current MCS module is increased by 2 KB but the memory of the successor is decreased by 2 KB.

The layout configuration BORROW is borrowing the 4 KB memory page of the successive MCS instance for the current instance. Thus, the memory of the current MCS module is increased by 4 KB but the memory of the successor is decreased by 4 KB.

It should be noted that the successor of the last MCS instance is the first MCS instance MCS0.

The actual size of the memory pages for an MCS instance depends on the layout configuration for the current instance MCS[i] and the layout configuration of the preceding memory instance MCS[i-1].

3.2 Scheduling

The MCS sub module provides two different scheduling schemes: *round-robin schedule* and *accelerated schedule*.

The *round-robin* order scheduling assigns all MCS-channels an equal amount of time slices.

In addition, the scheduler also assigns one time slice to the CPU, in order to guarantee at least one memory access by the CPU within each round-trip cycle.

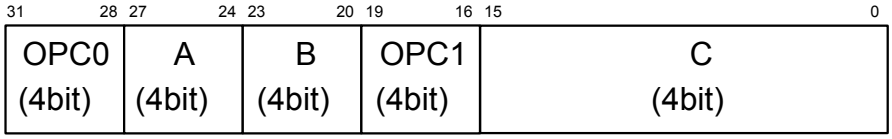
The second scheduling mode, the *accelerated scheduling mode*, improves the computational performance of the MCS by skipping suspended MCS-channels (and channels that are currently in stage 0, 1, 2, or 3) during scheduling.

In summary, the *round-robin scheduling* mode grants time slices of suspended MCS channels to the CPU and the *accelerated scheduling* mode grants time slices of suspended MCS-channels to non-suspended MCS-channels.

3.3 Instruction format

In general, each instruction is 32 bit wide but the duration of each instruction varies between several *instruction cycles*.

Figure 4. MCS opcode



Common used terms, abbreviations:

- **OREG**: the operation register set OREG = {R0, R1, R2...R7, STA, ACB, CTRG, STRG, TBU_TS0, TBU_TS1, TBU_TS2, MHB}
- **AREG**: the ARU register set AREG = {R0, R1, R2,..., R7, ZERO}
- **LIT4**: the set LIT4 = {0,1...15} is an unsigned 4-bit literal
- **LIT16**: the set LIT16 = {0,1...216-1} is an unsigned 16-bit literal
- **LIT24**: the set LIT24 = {0,1...224-1} is an unsigned 24-bit literal
- **MEMORY ADRESSING**: the expression MEM(X) represents the 32-bit value at address X of the memory
- **ARU ADRESSING**: in the case of ARU reading, the expression ARU(X) represents the 53-bit ARU word of ARU channel at address X

Example 1 : The instruction: **MOVL A, C** where A is an OREG and C is a literal.

4 MCS programming

The behavior of the different MCS tasks for a single MCS module can be specified in one MCS assembler source file, referred as MCS file throughout the document, which typically has the file extension *.mcs*.

ASM-MCS translates this file into machine code that is loaded into the dedicated MCS RAM module. The generated machine code can be represented as a C-Code Array that can be processed by a C-Compiler.

An MCS file typically consists of a list of MCS instructions mixed up with several assembler directives, which are not directly translated into machine code, but they are used to tell the tool ASM-MCS how to translate the instructions. The line comment are identified by the character semicolon (;) or the character hash (#).

Each instruction of an MCS file corresponds to the following form:

MNEMONIC [<ARG1> [,] <ARG2> … [,] <ARGN>],

whereas <MNEMONIC> is a string that identifies the instruction to be coded.

ASM-MCS provides a set of so called assembler directives that are controlling the assembling procedure:

- **.include "<INCFILE>"**

  The INCLUDE directive can be used to insert the content of another MCS file in the current MCS file.
- **.org <EXPR>**

  The ORG directive can be used to manipulate the address counter of ASM-MCS in order to control the memory regions in which ASM-MCS will put the assembled code or to reserve data sections in the memory to store program variables.
- **.define <VARNAME> <EXPR>**

  The DEFINE directive can be used to create assembler variables.
- **<LABEL>**

  A LABEL directive can be used to create symbolic identifiers and map the current value of the address counter to this symbol.
- **.register <REGNAME> <EXPR>**

  A REGISTER directive enables the creation of symbolic names for registers.
- **.var <EXPR> [<WIDTH>]**

  The VARIABLE directive provides a possibility to initialize program variables that are located in the MCS memory.

4.1 MCS template program

The MCS program can be divided in different sections:

- Include section.
  - *.include "mcs24_2.inc"*
- Definition of variable and custom label (#define).
  - *.define EN_L_MSK 0xFFFFFE*
- Reset vector (start of the MCS program).
  - *.org 0x0*
    jmp tsk0_init
- Stack definition (if need, not mandatory).
  - *tsk0_stack:*
    .org (tsk0_stack+0x40)
- Source code for each channel.
  - *tsk0_init:*
    movl R7 (tsk0_stack-4); initialize stack pointer of task 0
    movl R0, 0
    …..

Figure 5. **MCS example file**

```
; Prepare assembler for MCS memory
;
.include "mcs24_2.inc"                ; This is specific for the current archietcture specification
.define EN_L_MSK  0xFFFFFE
; Initialize reset vectors of MCS 0
;
.org 0x0
jmp tsk0_init
jmp tsk1_init
jmp tsk2_init
jmp tsk3_init
jmp tsk4_init
jmp tsk5_init
jmp tsk6_init
jmp tsk7_init

; Allocate stack frames (e.g., allocate 16 words memory locations)
;
tsk0_stack:
.org (tsk0_stack+0x40)
tsk1_stack:
.org (tsk1_stack+0x40)
….
…..
tsk7_stack:
.org (tsk7_stack+0x40)

; Program entry for MCS-channel 0 - task0
;
tsk0_init:
  movl R7 (tsk0_stack-4) ;initialize stack pointer of task 0
  movl R0, 0
….
  movl R6, 0

TASK0_START:
  movl CTRG, 1                         ;Clear start trigger

TASK0_WAIT_TRIGGER_START:
  movl  R6, 1
  wurm  R6, STRG, 1
  movl CTRG, 1                         ;Clear start trigger

TASK0_RUN:
  ;insert here the MCS user code

TASK0_JUMP_TO_NEXT_UPDATE:
  jmp   TASK0_START

TASK0_END:
;Should not get here
          andl STA EN_L_MSK            ;Disable MCS channel 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Unused MCS0 Threads
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
tsk1_init:
   andl STA EN_L_MSK1 ; disable MCS

tsk2_init:
    andl STA EN_L_MSK2 ; disable MCS
….
….
tsk7_init:
          andl STA EN_L_MSK7 ; disable MCS
```

4.2 How to integrate MCS code in your application

MCS code can be integrate in a C project generating MCS opcodes and linking the opcode structure in the project source file.

Assembly tool is available to generate C code, with the header file where all the labels are defined as offset staring from the root of the MCS memory.

Starting from mcs file use the line:

asm-mcs.exe -o out_file.c -log log_file -odef out_file.h -olbl mcs0_mem -I source_dir -I <mcs assembly source directory> soucefile.mcs

This command starting e.g. from mcs0.mcs generates two different files:

- mcs0.c (out_file.c)
- mcs0.h (out_file.h)

Figure 6. MCS C code

```
/* generated by MCS-Assembler tool ASM-MCS version 0.9 */
/* Copyright (C) 2011-2016 by Robert Bosch GmbH, Germany */
/* target architecture : mcs24-1 */

unsigned long mcs0_mem[61] = {
    0xE00000A4,
    0xE00000F0,
    0xE00000F0,
    0xE00000F0,
    ….
};
```

Figure 7. MCS header file

```
/* generated by MCS-Assembler tool ASM-MCS version 0.9 */
/* Copyright (C) 2011-2016 by Robert Bosch GmbH, Germany */
/* target architecture : mcs24-1 */

#ifndef MCS0_H_
#define MCS0_H_

#define OFFSET_MCS0_MEM     (   0) /* byte address offset for assembled
                                      code in array C-array 'mcs0_mem' */
#define SIZE_MCS0_MEM       ( 244) /* code size in bytes of assembled
                                      code in C-array 'mcs0_mem' */

#define LABEL_MCS0_MEM_TSK0_STACK  (  25) /* Index into C-array
                                    'mcs0_mem' for assembler label 'TSK0_STACK' */
```

Add C file in the list of source file to build in your project, it will be linked in the final binary file.

Label (in the header file) can be referenced in the application source code.

5 PWM signal using MCS

PWM signal can be generated by MCS IP. Typically its usage is useful in the Engine actuation using the MCS capabilities in conjunction with the other GTM-IP, like DPLL, TIM events, etc.

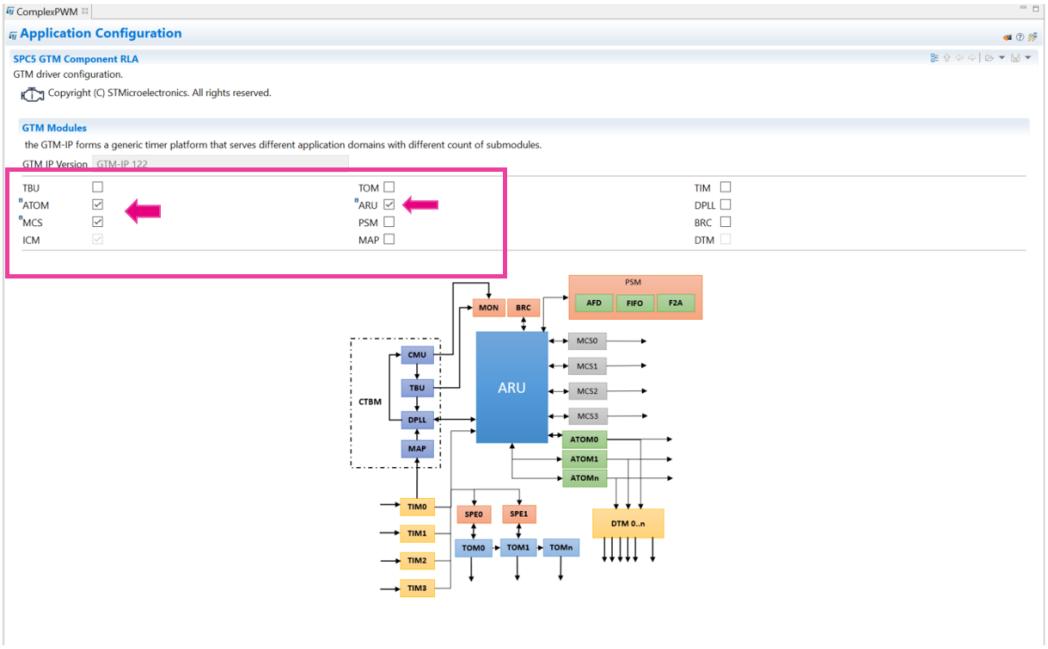
In this example a PWM signal will be generated by MCS. Period, duty and clock source are managed by MCS itself. CPU will trigger only the start.

In a real scenario the CPU trigger can be a TIM event trigger linked to a crank-wheel (for example).

5.1 MCS setup

MCS will be setup using the SPC5Studio MCS IDE.

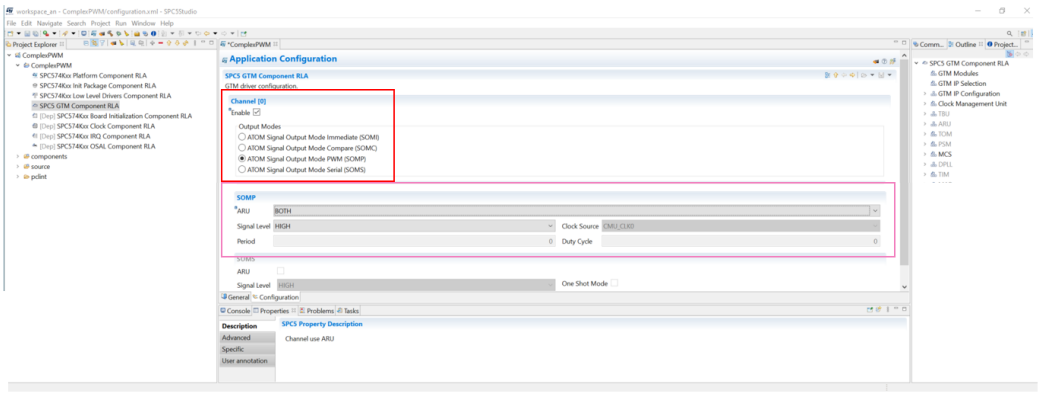
Figure 8. GTM config



PWM will be generated using ATOM0 Channel0.

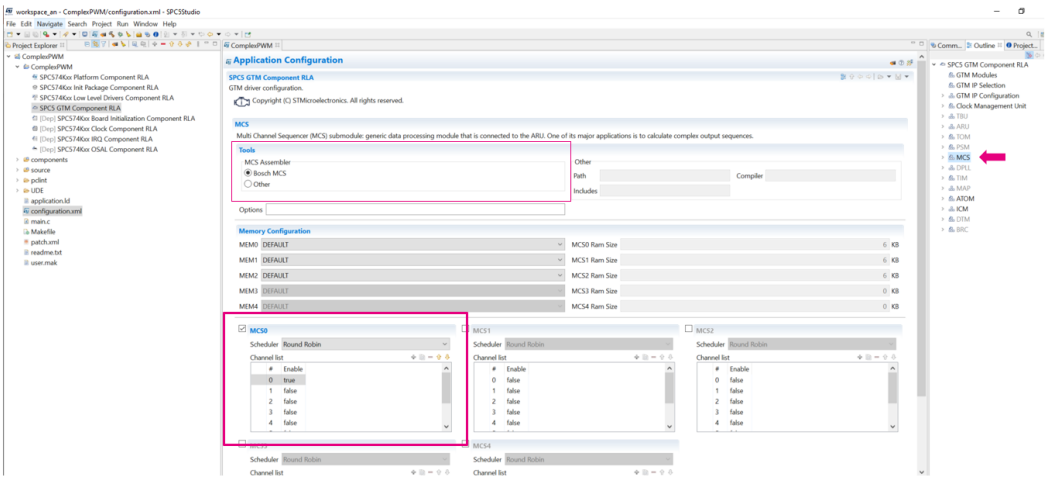
The Atom channel will be configured in SOMP mode, input values (BOTH) will be set by MCS via ARU.

Figure 9. ATOM setting



The PIN used is PD[14] with functionality: ATOM0_0
Enable MCS0 Channel0 using the round robin scheduler and default configuration memory.

Figure 10. MCS setting



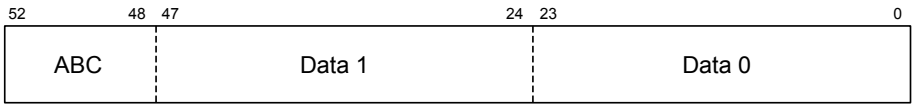
5.1.1 ARU (Advanced Routing Unit)

One central concept 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.

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.

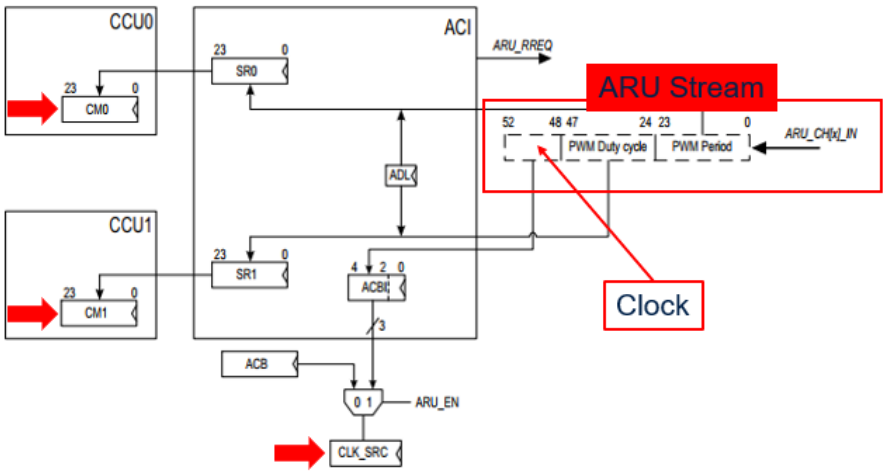
A connection between a data source and a data destination is also called ***data stream***.

Figure 11. ARU data stream



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

Figure 12. ARU data input stream for PWM



5.2 Code

5.2.1 Initialization C code

In the main application (*main.c*) must be defined the initialization APIs for the module used,

In SPC5Studio the GTM is automatically initialized in the *componentsInit()*, otherwise refer to the Application Note AN5552 (see Table 3) for a manual initialization.

After the GTM initialization, add the API specific for the MCS and PWM ATOM signal configuration.

A minimal main application:

Figure 13. **Main.c**

```
int main(void) {
/* Initialization of all the imported components in the order specified
   In the application wizard. The function is generated automatically. */
    componentsInit();

    /* Enable Interrupts */
    irqIsrEnable();

    /* Enable Clock Management Unit */
    gtm_cmuStart(&CMUD1);

    /* Define the source trigger for the ATOM0 CHANNEL0, in this case
      the port 0 of MCS0 is used as ARU interface (24 port for each
      MCS are available)
     */
    gtm_atomSetDataSource(&ATOMD1, ATOM_CHANNEL0,
                          SPC5_GTM_WRITE_ADDRESS_MCS0_MCS0_0);

    gtm_atomStart(&ATOMD1, ATOM_CHANNEL0);

    /* Initialize the MCS */
    initMCS();

    /* PWM Duty, Period and Clock are set in the MCS code.
     * MCS code execution start when one of the Trigger bit is set
     * in this example, bit "0" is used. */
    gtm_mcsSetTriggerBit(&MCSD1, MCS_CHANNEL0);

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

     }
```

Figure 14. **Auxiliary MCS routine**

```
/*
 * Initializes the MCS.
 */
static void initMCS(void) {

      /* Reset the RAM and wait for action complete */
      gtm_mcsResetRAM(&MCSD1);
      while (gtm_mcsGetResetRAM(&MCSD1) == 1U) {
            ;
      }
      /* Load the MCS program
       * These symbols are available in the mcs0.c and mcs0.h generated files
       */
      gtm_mcsLoadProgram(&MCSD1, mcs0_mem, SIZE_MCS0_MEM, OFFSET_MCS0_MEM);

      /* Enable the channel */
      gtm_mcsEnableChannel(&MCSD1, MCS_CHANNEL0);

       }
```

5.2.2 MCS code

MCS file (mcs0.mcs) will be parser from the MCS assembly tool to generate mcs0.c and mcs0.h. These generated files can be added to the compiler list objects.

Figure 15. **MCS code mcs0.mcs (1 of 2)**

```
; 1) prepare assembler for MCS memory
; -------------------------------------
.include "mcs24_2.inc"

; 2) define some constants
; ------------------------
.define EN_L_MSK $FFFFE

.define PWM_PERIOD            1000
.define PMW_DUTY               500
.define PWM_CLK_SRC            0x0
.define PWM_MCS0_0_PORT_IDX     $0

; 3) initialize reset vectors of MCS channels 0
; ----------------------------------------------------
.org 0x0
jmp tsk0_init
jmp tsk1_init
jmp tsk2_init
jmp tsk3_init
jmp tsk4_init
jmp tsk5_init
jmp tsk6_init
jmp tsk7_init

.org 100
tsk0_stack:
  .org (tsk0_stack+0x40)
```

Note: MCS code continue in next figure:

Figure 16. **MCS code mcs0.mcs (2 of 2)**

```
;User code for the Channel0
tsk0_init:
  movl R7 (tsk0_stack-4)                    ; initialize stack pointer of task 0
  movl R0, 0
  movl R1, 0
  movl R2, 0
  movl R3, 0
  movl R4, 0
  movl R5, 0
  movl R6, 0

TASK0_START:
  movl CTRG, 1                               ; Clear start trigger

TASK0_CHECK_START_SEQUENCE:
  movl  R0, 1
  wurm  R0, STRG, 1
  movl CTRG, 1                              ; Clear start trigger

;Configure PWM
  movl ACB, PWM_CLK_SRC
  movl R1, PMW_DUTY
  movl R0, PWM_PERIOD
  awr R0, R1, PWM_MCS0_0_PORT_IDX           ; Set PWM parameter for ATOM
                                            ; from this instruction PWM signal
                                            ; Start.

  xorl STA, 2                               ; raise ISR

TASK0_JUMP_TO_NEXT_UPDATE:
   jmp   TASK0_CHECK_START_SEQUENCE

TASK0_END:
       andl STA EN_L_MSK                  ; disable MCS channel 0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Unused MCS0 Threads
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
tsk1_init:
tsk2_init:
tsk3_init:
tsk4_init:
tsk5_init:
tsk6_init:
tsk7_init:

   andl STA EN_L_MSK ; disable MCS
```

For more details about the assembly instruction, refer to the Reference Manual RM0361 (see Table 3).

The instruction *"awr R0, R1, PWM_MCS0_0_PORT_IDX"*, ARU Write instruction, set the ATOM Shadow register with PWM parameter. In the ARU stream (53 bit length) PWM period is available in the low 24 bit (0..23),PWM duty in high bit (24..47), clock source defined in the ACB bit (48..52).

When this instruction is triggered by MCS code the PWM starts its execution.

In the register R0 have been set the prid, in R1 the duty, in ACB register the clock.

The macro *PWM_MCS0_0_PORT_IDX* define the ARU port used as source resource for the ATOM, a match of ARU index/address is mandatory between the IP that receive settings and the IP that require setting. In this case MCS0 and ATOM0 Channel0

In the MCS code:

.define PWM_MCS0_0_PORT_IDX 0

ATOM (main.c) code:

gtm_atomSetDataSource(&ATOMD1, ATOM_CHANNEL0,
***SPC5_GTM_WRITE_ADDRESS_MCS0_MCS0_0**);*

The ***"SPC5_GTM_WRITE_ADDRESS_MCS0_MCS0_0"*** is the ARU address of MCS0 port 0.

List of all ARU address is available in aru.h file.

Figure 17. PWM signal

The PWM signal is generated using the CMU_CLK0 clock with output frequency of 1 MHz.

Duty set to 50%, Period set to 1000 tick.

Using these settings, the PWM have an output freq. of 1 KHz, T = 1 ms.

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

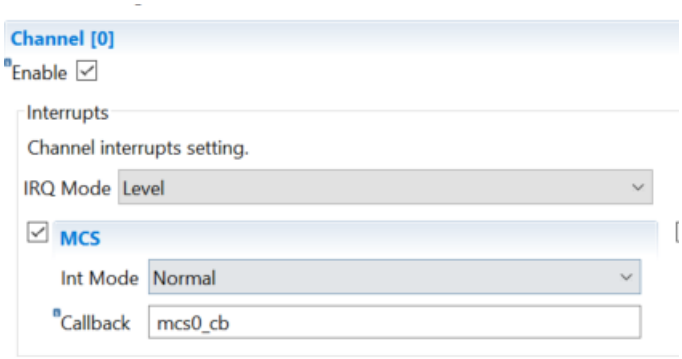
$$f_{out} = \frac{1000000}{1000} = 1\ KHz \quad (2)$$

$$T_{out} = \frac{1}{1\ KHz} = 1\ ms \quad (3)$$

5.2.3 MCS interrupt

MCS give the possibility to raise interrupts to better control the application and take the appropriate action.
The interrupt raises toward CPU setting the bit 2 of MCSx STATUS register.
MCS instruction: ***xorl STA 0x2*** *(or movl STA 0x3)*
Define the MCS call back in the SPC5Studio GTM-MCS IDE

Figure 18. MCS call back definition



Define the call back in your main application:

Figure 19. MCS call back code

```
void mcs0_cb(GTM_MCSDriver *mcsd, uint8_t channel) {
      (void)mcsd;
      (void)channel;

      //Your code
}
```

6 Conclusion

The document gives all the information and method to program your application using the GTM MCS code. As example has been used the generation of a Complex PWM signals using the tool SPC5Studio but also the needed code to integrate a PWM project outside the tool.

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

A set of APIs is available to be used in the user code to customize the GUI configuration parameters and the application development.

The online documentation is available to help the user in his project application development.

Appendix A Reference documents

Table 3. **Reference documents**

Doc Name	ID	Title
AN5552	034599	Generate PWM signals with GTM
RM0361	025070	Generic Timer Module specification revision 1.5.5.1

Revision history

Table 4. **Document revision history**

Date	Version	Changes
09-Sep-2021	1	Initial release.

Contents

List of tables

List of figures