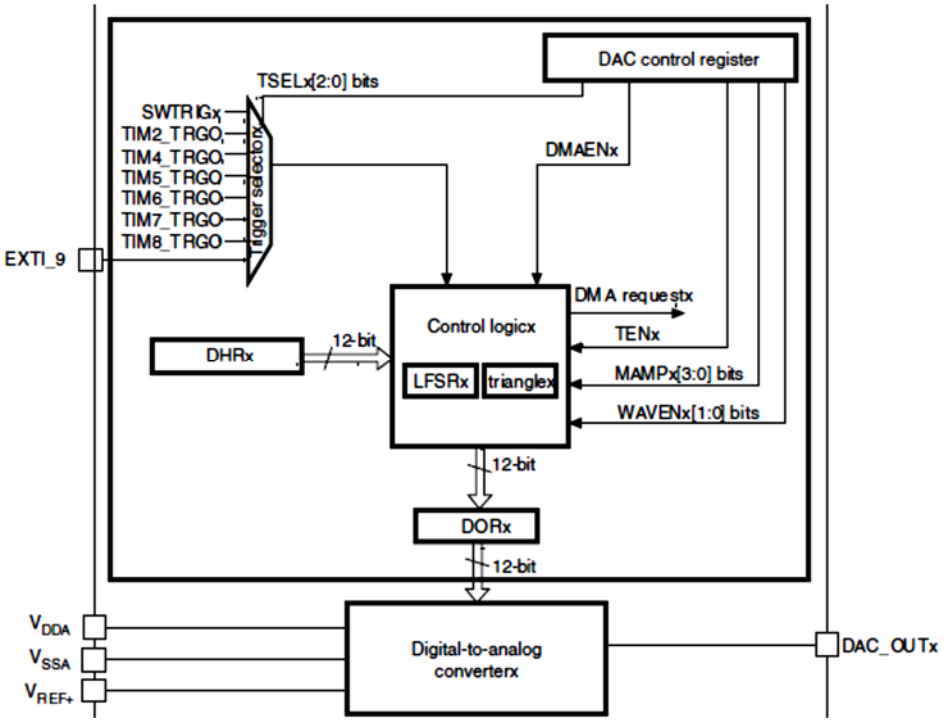
CONVERTISSEUR NUMERIQUE-ANALOGIQUE (DAC)

1. Introduction

Ce module comporte deux convertisseurs numériques analogiques ayant deux sorties indépendantes PA4/DAC1 et PA5/DAC2. Les deux canaux peuvent être configurés en mode 8 bits ou 12 bits et les conversions peuvent être effectués indépendamment ou simultané. La mise à jour de la sortie peut être synchronisée par des événements externes (par des timers). De plus, il est possible d'ajouter un pseud-bruit ou un signal triangulaire. Le schéma bloc du DAC est donné par la figure suivante :



Ce convertisseur comporte trois registres principaux :

- Le registre DHRx (Data Holding Register), en réalité se registre est représenté par une combinaison de registres DHR8Rx, HHR12Rx et DHR12Lx, pour permettre la configuration en mode 8bits ou 12 bits alignés à droite ou à gauche.
- Le registre DORx (Data Out Register), ce registre est chargé par le contenu du registre DHRx. Il y deux modes de chargement de ce registre : automatiquement après l'écriture dans DHRx ou après l'arrivée d'un événement matériel.
- Le registre de contrôle CR (Control Register), ce registre permet la validation du DAC, l'ajout d'un bruit ou signal triangle, mise à jour par événement matériel ou logiciel et utilisation du DMA ou non.

2. Programmation

Nous Nous intéressant à une utilisation simple ; fonctionnement en mode indépendant avec mise à jour automatique de registre DORx.

Les convertisseurs DAC1 et DAC2 sont connectés respectivement aux lignes PA4 et PA5. Donc il faut configurer ces broches en analogiques si on veut les associes aux DAC.

Déclarer une structure de type *DAC_InitTypeDef*

```
typedef struct
{
  uint32_t DAC_Trigger;  /* événement de mise à jour du registre DORx :
                            DAC_Trigger_None  : juste après écriture dans DHRx
                            DAC_Trigger_T2_TRGO : événement Timer 2
                            DAC_Trigger_T4_TRGO : événement Timer 4
                            DAC_Trigger_T5_TRGO : événement Timer 5
                            DAC_Trigger_T6_TRGO : événement Timer 6
                            DAC_Trigger_T7_TRGO : événement Timer 7
                            DAC_Trigger_T8_TRGO : événement Timer 8
                            DAC_Trigger_Ext_IT9 : événement EXTI_linge9
                            DAC_Trigger_Software : logicielle   */

  uint32_t DAC_WaveGeneration;  /* génération d'une onde additionnelle
                            DAC_WaveGeneration_None : contenu de DHRx seul
                            DAC_WaveGeneration_Noise : ajout d'un bruit
                            DAC_WaveGeneration_Triangle : ajout d'un signal
                                                          Triangle */
  uint32_t DAC_LFSRUnmask_TriangleAmplitude; /* define l'amplitude de l'onde
     si on a choisi un bruit ou une onde triangle.ce paramètres prend l'une des
     valeurs suivantes :
                        DAC_LFSRUnmask_Bits0
                        DAC_LFSRUnmask_Bits1_0
                        DAC_LFSRUnmask_Bits2_0
                        DAC_LFSRUnmask_Bits3_0
                        DAC_LFSRUnmask_Bits4_0
                        DAC_LFSRUnmask_Bits5_0
                        DAC_LFSRUnmask_Bits6_0
                        DAC_LFSRUnmask_Bits7_0
                        DAC_LFSRUnmask_Bits8_0
                        DAC_LFSRUnmask_Bits9_0
                        DAC_LFSRUnmask_Bits10_0
                        DAC_LFSRUnmask_Bits11_0
                        DAC_TriangleAmplitude_1
                        DAC_TriangleAmplitude_3
                        DAC_TriangleAmplitude_7
                        DAC_TriangleAmplitude_15
                        DAC_TriangleAmplitude_31
                        DAC_TriangleAmplitude_63
                        DAC_TriangleAmplitude_127
                        DAC_TriangleAmplitude_255
                        DAC_TriangleAmplitude_511
                        DAC_TriangleAmplitude_1023
                        DAC_TriangleAmplitude_2047
                        DAC_TriangleAmplitude_4095   */
  uint32_t DAC_OutputBuffer;   /* ajout d'une adaptation d'impédance ou non :
                        DAC_OutputBuffer_Enable
                        DAC_OutputBuffer_disable
}DAC_InitTypeDef;
```

Invoquer les fonctions :

```
void DAC_Init(uint32_t DAC_Channel, DAC_InitTypeDef* DAC_InitStruct)
```

paramètres : DAC_Channel : DAC_Channel_1 ou DAC_Channel_2

DAC_InitStruct : DAC_InitStructure.

```
void DAC_Cmd(uint32_t DAC_Channel, FunctionalState NewState)
```

paramètres : DAC_Channel : DAC_Channel_1 ou DAC_Channel_2

NewState : ENABLE ou DISANLE

Ecriture d'une valeur dans le registre DHRx :

```
void DAC_SetChannel1Data(uint32_t DAC_Align, uint16_t Data) ou
void DAC_SetChannel2Data(uint32_t DAC_Align, uint16_t Data)
```

paramètres : DAC_Align : DAC_Align_12b_R ou DAC_Align_12b_L ou DAC_Align_8b_R

Data : valeur entre 0 .. 4095 max.

3. Configuration de DAC1

```
GPIO_InitTypeDef GPIO_InitStructure;
DAC_InitTypeDef  DAC_InitStructure;

void DAC_Ch1_Config(void)
{
  /* GPIOA Periph clock enable */
  RCC_APB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

  /* DAC channel 1  (DAC_OUT1 = PA.4) configuration */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* DAC Periph clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
  /* DAC channel1 Configuration */
  DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
  DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
  DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
  DAC_Init(DAC_Channel_1, &DAC_InitStructure);
  /* Enable DAC Channel1 */
  DAC_Cmd(DAC_Channel_1, ENABLE);
}
```