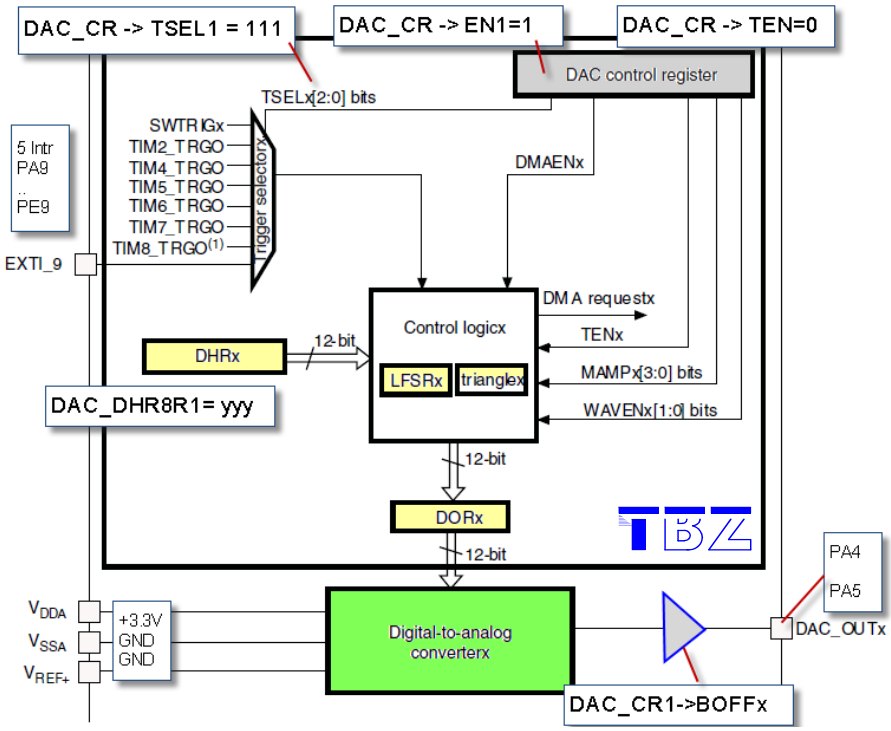
MCB32 APPN
DA Wandler mit MCB32



MCB32 - Embedded Programmierung

Version: **1.1521a**

Bitte beachten. Diese Unterlagen können ohne Vorankündigung jederzeit angepasst, verbessert und erweitert werden. Wir bitten Sie Wünsche und auch Fehler zu melden. (info@mcb32.ch)

1 Inhaltsverzeichnis

3 Einleitung Beschreibung Wandler

3.1 DAC * 12bit

[1, p. Kap. 12.1/ p253]*"The DAC module is a 12-bit, voltage output digital-to-analog converter. The DAC can be configured in 8- or 12-bit mode and may be used in conjunction with the DMA controller. In12-bit mode, the data could be left- or right-aligned. The DAC has two output channels, each with its own converter. In dual DAC channel mode, conversions could be done independently or simultaneously when both channels are grouped together for synchronous update operation. An input reference pin VREF+ (shared with ADC) is available for better resolution."*

Der STM32F107 hat 2 Stück 12-Bit DAC. Eine eingebaute Logik kann zudem Dreiecksignale oder Rauschen erzeugen. Die 2 DAC können unabhängig voneinander oder synchron betrieben werden. Eine eingebaute Referenzspannung erlaubt einen unabhängigen Betrieb. Ein zuschaltbarer Ausgangsbuffer wird mit dem BOFFx-Bit ein- resp. ausgeschaltet. Das Ausgangsformat kann 8Bit resp. 12Bit sein. Im 12Bit Modus können die Bits links oder rechts justiert werden. (left- / right alignment). Der DAC liefert sein Ausgangssignal entweder von externen Triggern oder dann mittels eines SW-Triggers. [1, p. Tab.74]

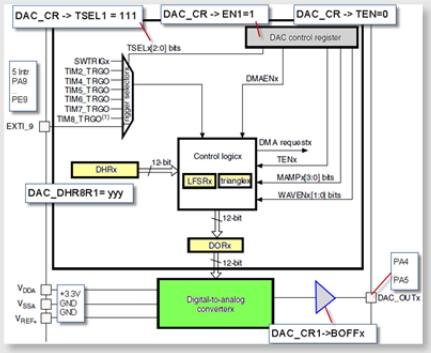


Abbildung 1: Auszug aus DAC-Blockdiagram

3.1.1 Berechnung der DAC- Ausgangsspannung

$$DACOutput = V_{Ref} * \frac{DOR}{4095}$$

Ohne externen Trigger startet die Wandlung durch Eintrag des Digitalwertes in ein Register von:

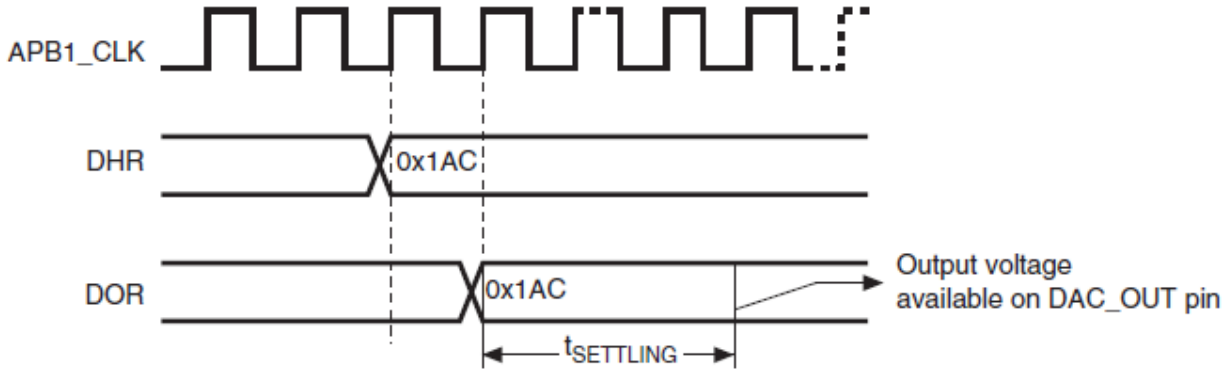
- DAC_DHR8R1 8 Bit
- DAC_DHR12R1 12 Bit rechtsbündig
- DAC_DHR12L1 12 Bit linksbündig

und automatischer Übergabe ins Data Output Register DORx.

3.1.2 Trigger

Der DAC arbeitet mit einem sogenannten Schattenregister (Shadow-Register), welches die Werte aus dem Datenregister wahlweise erst nach einem Trigger Impuls übernimmt. Als Trigger können verschiedene Zeitgeber (Timer), EXTI Line 9, Software oder keiner ausgewählt werden. In letzterem Fall werden die Werte sofort übernommen.

3.1.3 Timing Diagramm für Wandlung mit Trigger ausgeschaltet TEN = 0 [1, p. Kap.12.3.xx]



3.2 DAC Pin Beschreibung [1]

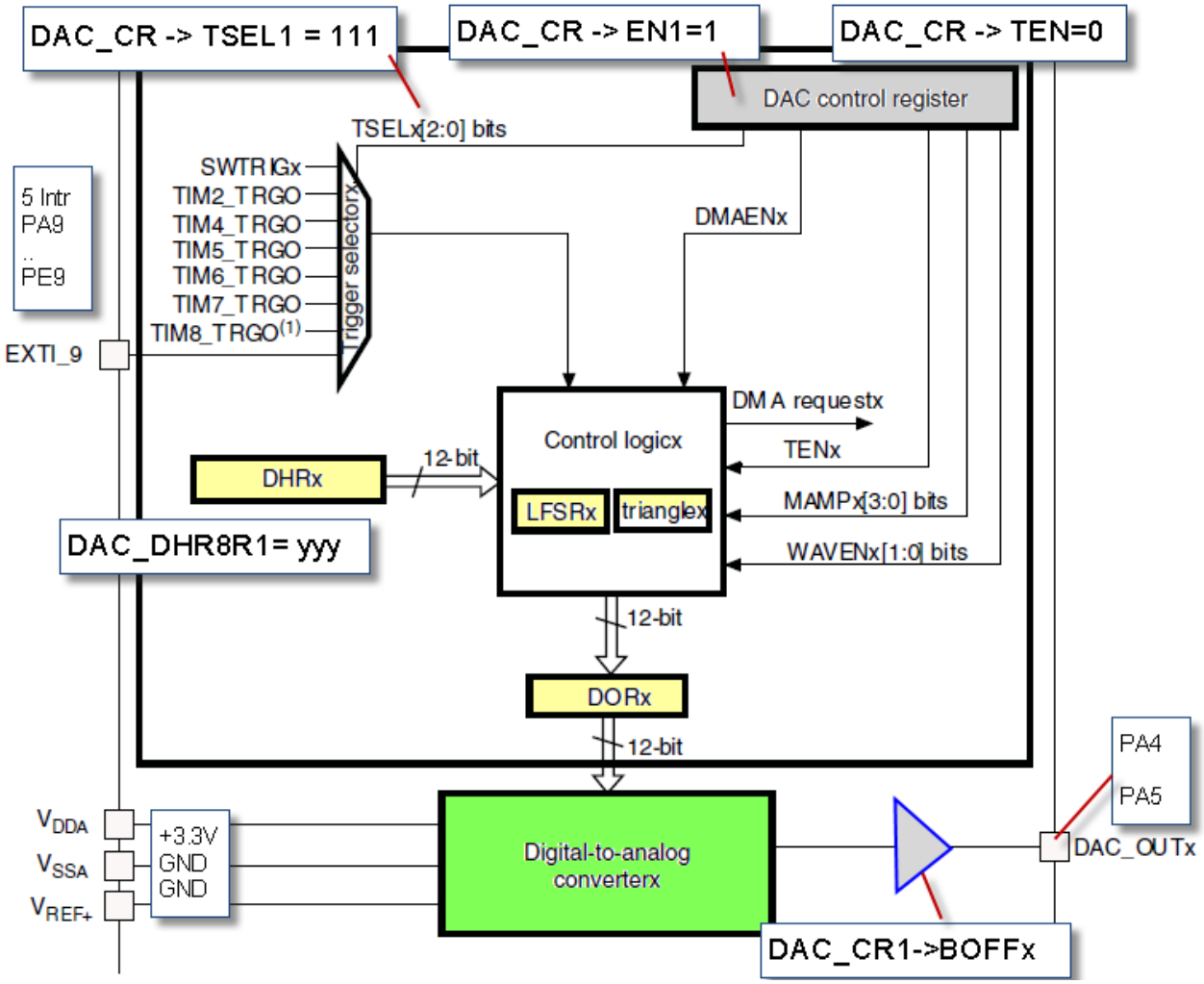
Die Pins des DAC arbeiten innerhalb eines bestimmten Arbeitsbereiches. Dieser Bereich hängt von der angelegten Speise- und Referenzspannung ab

1. Technische Daten eines DAC Pins (DAC_Outx : x=1 / 2) [1, p. 254 / Tab 73]

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the DAC, 2.4 V ≤ V_{REF+} ≤ V_{DDA} (3.3 V)
V_{DDA}	Input, analog supply	Analog power supply
V_{SSA}	Input, analog supply ground	Ground for analog power supply
DAC_OUTx	Analog output signal	DAC channelx analog output

Abbildung 2: ADC Pins, Spannungsbereiche (Table 65) für STM32F107 Chip mit 16 Kanälen

2. Abbildung 3: DAC Blockschema von DAC1 sowie DAC2



-

1. Beispiel mit einfachem Setup:

Aufgabe: Wir wollen die Werte von P0 auf dem DAC1 (PA4) ausgeben

Um Nebeneffekte zu vermeiden (avoid parasitic consumption) ,müssen die beiden Ports PA4 und PA5 zuerst als analoge Pins konfiguriert sein.

Also setzen wir das Port Konfiguration-Register (GPIOC_CRL) entsprechend dem Manual (siehe [1, pp. Tabelle 73, Notiz)]

3.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

PA4 wird als Input gesetzt (siehe Manual).

GPIOA->CRL &= 0xFFFF0FFF;

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	0 rw	0 rw	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

CNFy[1:0]: Port x configuration bits (y= 0 .. 7)

00: Analog input mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved

00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

MODEy[1:0]: Port x mode bits (y= 0 .. 7)

00: Input mode (reset state)
ODRx = 0 bewirkt PullDown
ODRx = 1 bewirkt PullUp;

01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

3.2.2 APB1 peripheral clock enable register (RCC_APB1ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	CAN2 EN	CAN1 EN	Reserved		I2C2 EN	I2C1 EN	UART5E N	UART4E N	USART3 EN	USART2 EN	Res.
		rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw						rw	rw	rw	rw	rw	rw

Bit 29 **DACEN**: DAC interface clock enable

0: DAC interface clock disabled
1: DAC interface clock enable

Abbildung 4: Peripherial Clock Register

Bit 29 auf 1 setzen:. Siehe Bild oben. (**RCC->APB1ENR |= 1<<29**;) Damit ist der Clock eingeschaltet.

3.2.3 DAC control register (DAC_CR) [1, p. Kap. 12.5.1/p264]

Der DAC Kanal wird im nächsten Schritt mit Daten versorgt. Das heisst wir wählen die Triggerart und schalten den DAC ein.

Befehlssequenz: **DAC->CR = (0x7<<3)+1;**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			DMA EN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]			TEN2	BOFF2	EN2
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			DMA EN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

TSEL1[2:0]: DAC channel1 trigger selection

These bits select the external event used to trigger DAC channel1

000: Timer 6 TRGO event
001: Timer 3 TRGO event in connectivity line devices,
Timer 8 TRGO in high-density and XL-density devices
010: Timer 7 TRGO event
011: Timer 5 TRGO event
100: Timer 2 TRGO event
101: Timer 4 TRGO event
110: External line9
111: Software trigger

Note: only used if bit TEN1 = 1 (DAC channel1 trigger enabled)

2. DA Wandler Musterprogramme

.

Lösung1: DA Wandlung. 8Bit Port P0 wird ausgegeben

Das Pogramm liest den Port P0 und gibt die Daten via das Ausgaberegister DHR8R1 (8BitModus) aus. Das heisst die 8Bit vom Port steuern den vollen Bereich aus. Die Triggerart (SW) spielt keine Rolle da TEN=0. Der Wert wird immer ausgegeben sobald das Register DHRxxxx geladen wird.

```
#include "TouchP0P1.h"        // P0-,P1,Touchscreen
int main(void)
{
 InitTouchP0P1("1");

 GPIOA->CRL &= 0xFFFF0FFFF;   // PA4 analog IN für Out
 RCC->APB1ENR |= 1<<29;       // DAC Clock ON
 DAC->CR       = (0x7<<3)+1; // DAC1 SW-Trigger & Enable

 while(1)                     // Endlosschlaufe
 {
   DAC->DHR8R1 = P0;          // In P0 zu analog Pin PA4
 }
}
```

Abbildung 5: Musterprogramm für DA-Wandlung

Lösung2: 12Bit DA Wandler mit Loop

```
#include "TouchP0P1.h"
int main(void)
{
 long t, n;
 InitTouchP0P1("1");
                              // DAC1 initialisieren:
 GPIOA->CRL &= 0xFFFF0FFFF;   // PA4 analog IN für Out
 RCC->APB1ENR |= 1<<29;
 DAC->CR = (0x7<<3)+1;        // DAC1 SW-Trigger & Enable
 while(1)  // Endlosschlaufe
  {
   for(n=0;n<6;n++)           // 5 x 600mV Analogstufen
     {
      DAC->DHR12R1 = n*819;   // Analog 12 Bit-Auflösung
      delay_ms(1500);         // warte 1.5s (DVM)
      }
  }
}
```

Abbildung 6: Musterprogramm für DA Wandler mit Loop

Bei den obigen Programmen wird zwar der SW-Trigger gesetzt aber eigentlich nicht gebraucht. Dies, weil das BIT TEN1/2=0 gesetzt ist. Damit reagiert der DAC auf keinen Trigger sondern nur auf das Schreiben der DAC Register. Diese werden dann einen Zyklus später ausgegeben.

Folgendes Programm *DACTreppe_dual.c* korrigiert diesen Umstand:

Ausgabe von 5x ca. 600mv Treppensignal auf PA4/PA5 nach jeweils 1.5s *(Options C/C++: Optimize default, for Time)* mit Hilfe des SW-Triggers. Dabei werden beide DAC gleichzeitig geladen. Im Modus TENx=0 werden die DACs nacheinander geladen. Man beachte:

1. PA4 und PA5 werden einzeln konfiguriert. Das kann auch in einem Befehl passieren.
2. DAC1 und DAC2 werden mit einem Befehl enabled, auf externen Trigger geschaltet und der Trigger auf SW gestellt.
   `DAC->CR |= 0x003D003D;` [1, p. Kap. 12.5.1 / p264]
3. In der For-Schlaufe erhalten die DAC1/2 Register den zukünftigen Wert. Diese Werte gelangen mit dem SW-Trigger-Befehl:
   `DAC->SWTRIGR =0x3;` zum Ausgangsregister, dafür für beide DAC auf einmal. [1, p. Kap. 12.5.2 / p267]

Lösung3: DA Wandlung. gleichzeitig auf beiden Kanälen.

```
#include "TouchP0P1.h"

int main(void)
{
 long t, n;
 InitTouchP0P1("1");

 GPIOA->CRL &= 0xFFFF0FFFF;    // PA4 analog IN für Out
 GPIOA->CRL &= 0xFFF0FFFFF;    // PA5 analog IN für Out
 RCC->APB1ENR |= 1<<29;        // DAC Clock ON
//DAC->CR |= 0x00010001;       // DAC1/2 SW-Trigger & Enable für Version mit TENx=0
 DAC->CR |= 0x003D003D;  // DAC1/2 SW-Trigger & Enable für Version mit TENx=1
 while(1)
 {
 for(n=0;n<6;n++)
  {
   DAC->DHR12R1 = n*819;       // Analog 12 Bit-Aufloesung
   DAC->DHR12R2 = n*819;       // Analog 12 Bit-Aufloesung
   DAC->SWTRIGR =0x3;          // Wir haben den SWTRIGG Mod aktiv, TEN=1
                               // darum folgt an dieser Stelle der SW-Trigger
                               // Die beiden Reg. werden nun gleichzeitg
                               // ausgegeben.
   delay_ms(1500);             // warte 1.5s (DVM) sonst Zeit anpassen.
  } //for
 } //while
} //main
```

Abbildung 7: Musterprogramm für DA Wandler mit SW Trigger

Lösung4: 12Bit DA Wandler mit Rauschen und DC

```
#include "TouchP0P1.h"            // P0-,P1,Touchscreen

int main(void)                    // Hauptprogramm
{
 InitTouchP0P1("1");              // P0P1-Touchscreen ON

 GPIOA->CRL &= 0xFFFF0FFFF;       // PA4 analog IN für Out
 RCC->APB1ENR |= 1<<29;           // DAC Clock ON

 // 0xBBF = "Triangle Wave Generation" mit maximaler Amplitude
 // 0xB7F = "White Noise Generation"   mit maximaler Amplitude
 DAC->CR = 0xA7F;              // "White Noise Generation"   mit halber Amplitude

 while(1)
 {
   DAC->DHR12R1 = P0*16;      // P0 (12Bit) in DAC Ausgaberegister
   DAC->SWTRIGR=0x3;          // Addiert Noise zu DAC-
                              // Ausgaberegister und gibt alles aus.
 }
}
```

Abbildung 8: Musterprogramm für DA Wandler mit Rauschen und DC an PA4 (DAC1)

Nun wollen wir noch den internen Rauschgenerator testen. Dazu dient folgendes Programm. Wir erzeugen ein Rauschen mit halber Amplitude (FullScale/2) und addieren die via Port P0 erzeugte Spannung (DC) dazu. Das Programm kann anstelle des Rauschen auch ein Dreiecksignal überlagern.

Im Muster Code hat es weitere Beispiele welche zum Teil das richtige Vorgehen innerhalb der STM32 Programmierung via Strukturen benutzt. Obige Beispiele schreiben direkt in die Register des Prozessors. Damit ist der Einstieger nahe an der HW und wird gezwungen das Referenzmanual zu studieren [1].
Siehe folgendes Programm für Lösung4 mit Strukturen.

Das folgende Programm arbeitet wie die Lösung 4. Nun werde die Register aber nicht mehr direkt programmiert sondern via **Strukturen**.

```
//---------------------------------------------------
// DAC_DC_Noise_Strukturen.c
//---------------------------------------------------
//#include <stm32f10x.h>                      // Mikrocontrollertyp
#include "TouchP0P1.h"                        // P0-,P1,Touchscreen
int main(void)                                // Hauptprogramm
{
 GPIO_InitTypeDef GPIO_InitStructure;   // Eröffne Struktur für GPIO
 DAC_InitTypeDef  DAC_InitStructure;    // Eröffne Struktur für DAC

 // Clocks für DAC Betrieb initialisieren:
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
 // GPIO für DAC Betrieb initialisieren:
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
GPIO_Init(GPIOA, &GPIO_InitStructure);                  // DAC1 initialisieren:

// setze Amplitude auf Fullscale/2
 DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude = DAC_LFSRUnmask_Bits10_0 ;
 DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;  // Nutze den OutputBuffer (EIN)
 DAC_InitStructure.DAC_Trigger = DAC_Trigger_Software;          // Arbeite mit SW Trigger.
 DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_Noise ; //Noise, kein Dreieck
 DAC_Init(DAC_Channel_1, &DAC_InitStructure);            // Init nun für DAC1
 DAC_Init(DAC_Channel_2, &DAC_InitStructure);            //  "           DAC2

 DAC_Cmd(DAC_Channel_1, ENABLE);
 DAC_Cmd(DAC_Channel_2, ENABLE);

  InitTouchP0P1("1");                 // P0P1-Touchscreen ON
 delay_ms(3000);                      // Warte 3sek Messung vor dem Rauschen
 while(1)                             // Endlosschlaufe
 {
   DAC_SetDualChannelData(DAC_Align_12b_R, 0x0, (P0 * 16)); // Addiere P0 Wert zu Rauschen
   DAC_DualSoftwareTriggerCmd(ENABLE);       // Addiert Noise zu DAC-Ausgaberegister
                                             // und gibt alles aus.
 }
}
```

Abbildung 9: Musterprogramm für DA Wandler mit Rauschen und DC an PA4 (DAC1) mit Strukturen

5 Anhang Literaturverzeichnis und Links

[1] ST, «ARM_STM_Reference manual_V2014_REV15,» ST, 2014.
[2] J. Yiu, The definitive Guide to ARM Cortex-M3 and M4 Processors, 3 Hrsg., Bd. 1, Elsevier, Hrsg., Oxford: Elsevier, 2014.
[3] R. Jesse, Arm Cortex M3 Mikrocontroller. Einstieg und Praxis, 1 Hrsg., www.mitp.de, Hrsg., Heidelberg: Hütigh Jehle Rehm GmbH, 2014.
[4] Diller, «System Timer,» 06 07 2014. [Online]. Available: http://www.diller-technologies.de/stm32.html#system_timer. [Zugriff am 06 07 2014].
[5] A. Limited, «DDI0337E_cortex_m3_r1p1_trm.pdf,» ARM Limited, http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf, 2005, 2006 ARM Limited.
[6] A. C. Group, «http://www.vr.ncue.edu.tw/esa/b1011/CMSIS_Core.htm,» 2007. [Online].
[7] E. Malacarne, *Glossar Malacarne,* V11 Hrsg., Rüti: Cityline AG, 2014.
[8] R. Weber, «General Purpos Input Output,» 2014.
[9] R. Weber, «Projektvorlagen (div) MCB32,» 2013ff.
[10] E. F. E. Schellenberg, «Programmieren im Fach HST,» TBZ, Zürich, 2010ff.
[11] STM, «STM32F10x Standard Peripherals Firmware Library,» STM, 2010ff.
[12] ST, «STM32F107 Data Sheet_2014_REV7,» ST, 2014.

6 Anhang Wichtige Dokumente

Die folgende Liste zeigt auf die wichtigsten Dokumente welche im WEB zu finden sind. Beim Suchen lassen sich noch viele nützliche Links finden.

- Datenblatt (STM32F107VC) Beschreibung des konkreten Chips für Pinbelegung etc.
- Reference Manual (STM32F107VC) (>1000Seiten in Englisch)
  Ausführliche Beschreibung der Module einer Familie. Unter Umständen sind nicht alle Module im eingesetzten Chip vorhanden – siehe Datenblatt.
- Programming Manual (Cortex-M3)
  Enthält beispielsweise Informationen zum Interrupt Controller (NVIC).
- Standard Peripheral Library (STM32F10x)
  Im Gegensatz zu anderen MCUs sollen die Register der STM32 nicht direkt angesprochen werden. Dafür dienen die Funktionen der Standard Peripheral Library.
  Sie ist auf http://www.st.com/ zusammen mit einer Dokumentation (Datei: stm32f10x_stdperiph_lib_um.chm) herunterladbar.