CPE 323
MODULE 11
Digital-to-Analog Conversion

Aleksandar Milenković

Email: milenka@uah.edu

Web: http://www.ece.uah.edu/~milenka

Overview

This module introduces main concepts of analog-to-digital (AD) and digital-to-analog (DA) conversion. You will learn hardware aspects as well as software aspects of the analog-to-digital and digital-to-analog conversion and AD and DA converters. You will understand how to configure and utilize MSP430 ADC12 and DAC12 peripherals in your programs.

Objectives

- *Learners will understand hardware and software aspects of analog-to-digital and digital-to-analog converters*
- *Learners will understand how to configure and interact with MSP430 ADC peripheral*
- *Learners will understand how to configure and interact with MSP430 DAC peripheral*

Contents

1 DA Conversion: An Introduction 3
2 DA Conversion: A System View 4
3 DA Conversion: An Example 5
4 Digital-to-Analog Converter Types 7
4.1 Weighted Resistors Architecture 7
4.2 R-2R Ladder Network DAC 9
5 MSP430's DAC12 Controller 15
5.1 DAC12 Organization 16
5.2 DAC12 Control Registers 19
6 Code Example 22
7 Exercises 23

LaCASA

1 DA Conversion: An Introduction

Embedded computer systems are typically a part of other systems or devices. Four main tasks of any embedded computer system (or any computer system in general) are: (1) sensing the physical world (environment) using sensors; (2) processing information; (3) storing information; and (4) communicating information and acting on the environment. In this way, embedded systems are very similar to humans. Sensors or transducers are used to convert physical quantities (e.g., force, atmospheric pressure, sound, light, temperature, and others) into electrical signals (e.g., voltage or current) that we can measure. The electrical signals are often either noisy, weak, or both noisy and weak, so we rely on signal conditioning circuits to remove undesired harmonics of continual electrical signals (filtering) and amplify them (amplification) so they can be properly measured. Once the electrical signals are ready and in a desired range, a critical step is to convert them into corresponding digital values that can be further processed, stored, and/or communicated using digital computers. The process of converting analog electrical signals into binary numbers that correspond to the magnitude of the input signals is known as analog-to-digital conversion and is carried out using dedicated peripherals called analog-to-digital converters (ADCs).

One outcome of data processing is that we may need to act on the environment. For example, think about the air conditioning system at your house – a sensor continually measures the temperature (sensing), and if the temperature rises above a certain threshold (processing), a controller sends a signal to your AC unit to start pumping in cool air (acting on the environment). Once the temperature is lowered, the controller sends a signal to the AC unit to stop its operation. Acting on the environment sometimes requires that we generate analog electrical signals of certain amplitude and frequency. To generate such signals we conduct digital-to-analog conversion and for that we rely on peripheral devices called digital-to-analog converters (DACs).

In this module you will learn hardware aspects as well as software aspects of the analog-to-digital and digital-to-analog converters. You will understand how to configure and utilize MSP430 ADC and DAC peripherals in your programs. The very name MSP that stands for *Mixed Signal Processor* underscores the fact that MSP430 family of microcontrollers integrates a processor, non-volatile and volatile memories, and peripherals that deal with both analog and digital electrical signals.

`Things to remember 1-1. Analog-to-digital conversion.`

Analog-to-digital conversion is a process of converting analog continuous input signals (typically voltage or current) into discrete digital numbers that represent the magnitude of the input signal.

`Things to remember 1-2. Digital-to-analog conversion.`

Digital-to-analog conversion is a process of converting digital numbers into a continuous analog output signal.

2 DA Conversion: A System View

The process of converting digital values represented in binary into analog electrical signals is known as digital-to-analog conversion and is carried out using dedicated peripherals called digital-to-analog converters (DACs). Thus, DACs perform the reverse function of ADCs. DACs convert an abstract finite-precision number (integer or fixed-point binary number) into a physical quantity, typically an electrical signal. DACs are instrumental in many embedded applications. E.g., they are used in music players to convert digital data streams into analog audio signals, or in mobile devices and television to convert digital video data streams into analog video signals. Next, DACs are used in communications to create periodic sinusoidal signals.

Figure 1 shows a system view of digital-to-analog conversion. We provide a stream of digital data via N-bit DAC_DATA input lines and the DAC produces an analog output signal (V_{out}) with the magnitude proportional to the binary input as described in the equation below, where V_{REF} is the reference voltage.

$$V_{out} = V_{REF} \cdot \frac{DAC\_DATA}{2^N}$$

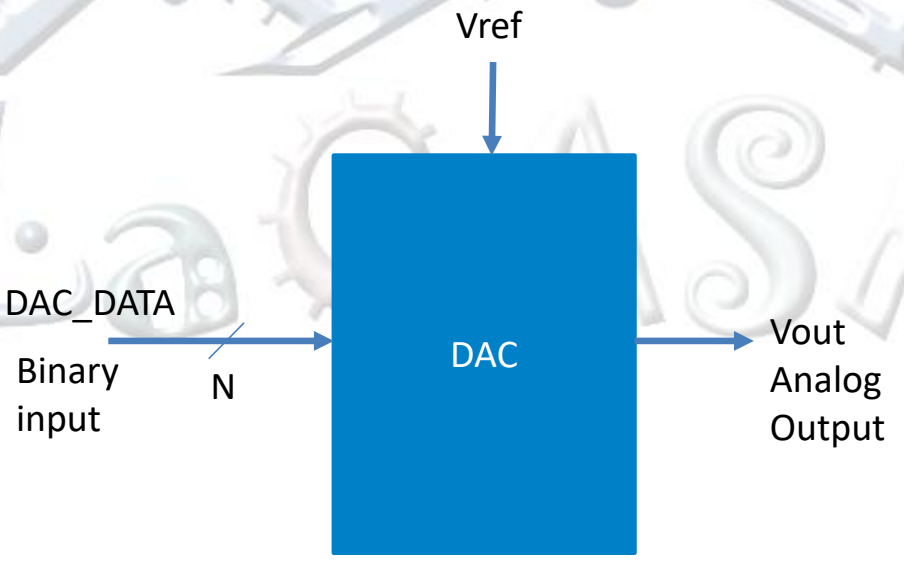


Figure 1. Digital-to-analog conversion: a system view.

Let us consider the following inputs. If we provide all zeros at the input, the output voltage is $V_{out} = 0$. If we provide all ones at the input, $V_{out} = V_{REF} \cdot \frac{2^N - 1}{2^N}$. Please note that the analog output never reaches V_{REF}. If we provide 1000..0b at the input, the output voltage is $V_{out} = \cdot \frac{V_{REF}}{2}$. Consequently, indeed this transfer function gives an output analog signal that is proportional to the binary input.

Things to remember 2-1. DAC Transfer Function.

The transfer function of an *N*-bit DAC with V_{REF} reference voltage producing analog output voltage V_{out} is expressed by the following equation:

$$V_{out} = V_{REF} \cdot \frac{DAC\_DATA}{2^N}$$

3 DA Conversion: An Example

Let us consider an analog signal, a_0, as shown in Figure 2. The ramp like signal is periodic with the period of $T_{a0} = 0.5\,ms$. The frequency of the input signal is $F_{a0} = 1/T_{a0} = 2{,}000\,Hz$. The signal rises from 0 to 2.5 V in 0.3 ms and falls back to 0 V in 0.2 ms. Please note that the signal is bounded between 0 and 2.5 V. Let us assume that our task is to create this ramp-like signal. Note: this example is a reverse of what we did in analog-to-digital conversion – here we have to send digital samples periodically to produce a signal that will look like the ideal ramp-like signal.

Two important questions related to the process of digital-to-analog conversion are as follows: (a) how many bits do we want in the binary representation (resolution of DA conversion); and (b) how many discrete samples do we want to send to the DAC in a unit of time (sampling frequency). By increasing the number of bits, we increase the resolution and accuracy of the DA conversion. By increasing the sampling frequency, we can create the signal closer to its ideal form.

Let us assume the following parameters of a DA conversion: the bit length of the binary representation is 8 bits and the number of samples per single period of the input signal is 5. This is equivalent to having 10,000 samples per second (2,000 Hz * 5 samples/period = 10,000 samples per second, or $F_s = 10{,}000$ sps). The sampling period is the the time distance between two consecutive samples and in this example is: $\Delta t_s = 1/F_s = 0.1\,ms$. The signal is bounded $0 \leq a_0 \leq 2.5\,V$, so we will set the reference voltage to $V_{REF} = 2.5\,V$.

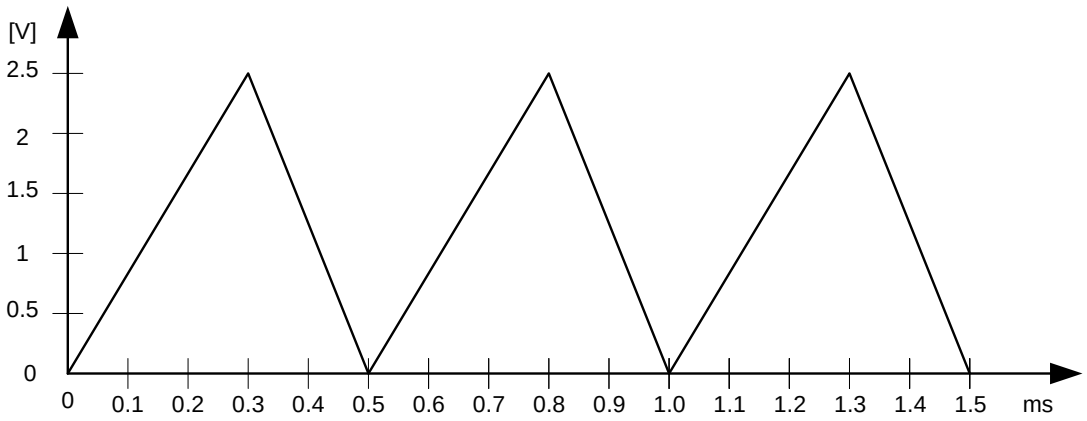


Figure 2. An example of the desired output analog signal.

We have already discussed how to determine the binary representation of samples taken at $t_0 = 0\ ms, t_1 = \Delta t_s = 0.1\ ms, t_2 = 2\Delta t_s = 0.2\ ms, t_3 = 3\Delta t_s = 0.3\ ms, t_4 = 4\Delta t_s = 0.4\ ms$. The digital values of these samples are 0, 85, 170, 255, 127 (or 128), respectively. How do we create the output analog signal then? A simple approach is that we create a lookup table in memory containing precomputed values of digital samples, e.g., `ramp_lt[5]={0, 85, 170, 255, 127}`. Next, we initialize a variable csample, that serves as an index in the lookup table, taking values, 0, 1, 2, 3, and 4. We can have a timer to generate a periodic interrupt request every 0.1 ms. Inside the timer's ISR we send the current sample to the DAC_DATA register of the DAC peripheral, DAC_DATA=`ramp_lt[csample]`, and increment the current sample index, `csample=(csample+1)%5`.

Figure 3 illustrates the output signal we create shown in red. It is overlapped with the ideal output signal given in black. The ramp shape of the desired signal is quite distorted and this is captured by the signal-to-noise ratio metric. One way to improve accuracy of the signal is to have more samples per one signal period. Repeat the exercise if $F_s = 20{,}000$ sps. Also, repeat the exercise assuming *N*=16 bits. How big lookup table would be in that case?

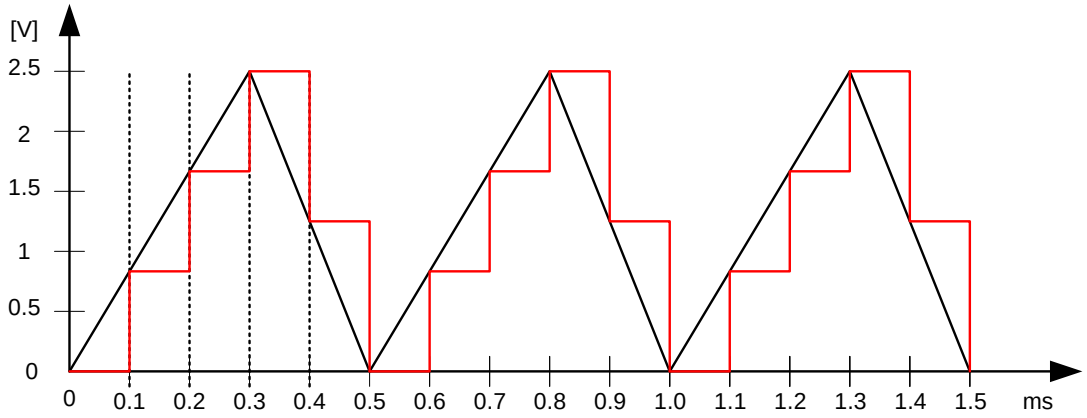


Figure 3. An example output analog signal created by a lookup table with 5 samples per period (one sample for every 0.1 ms).

The approach described above can be used for any type of output signal you want to create. In general, we are always limited with the number of bits in the binary representation as well as with how many samples we can acquire in any period of time, so our "digitized" interpretation of the real physical world is never perfect. The parameters to consider are size of each sample and the number of samples per period, impacting the the size of the lookup table. Please note that we do not have to have all samples pre-computed in the lookup table. For simpler shapes like in our example, the sample values can be computed in software every time you need to send them. However, for complex functions, e.g., sinusoidal signals, the computation can introduce significant overhead in processor time, so pre-computing the samples is preferred approach.

4 Digital-to-Analog Converter Types

DACs are simpler than ADCs to design. Here we will illustrate two implementations, one using weighted resistors and the other using the R-2R ladder network.

4.1 Weighted Resistors Architecture

Figure 4 shows schematic diagram of a 3-bit DAC using weighted resistors. It can easily be generalized to N-bit DAC. In the center we have an inverting summing amplifier. The binary input consists of 3 bits $b_2 b_1 b_0$, where b_0 is the least significant bit. Please note that these inputs control switches. E.g., if $b_0 = 0$, the leftmost switch connects to the ground, otherwise to $-V_R$.

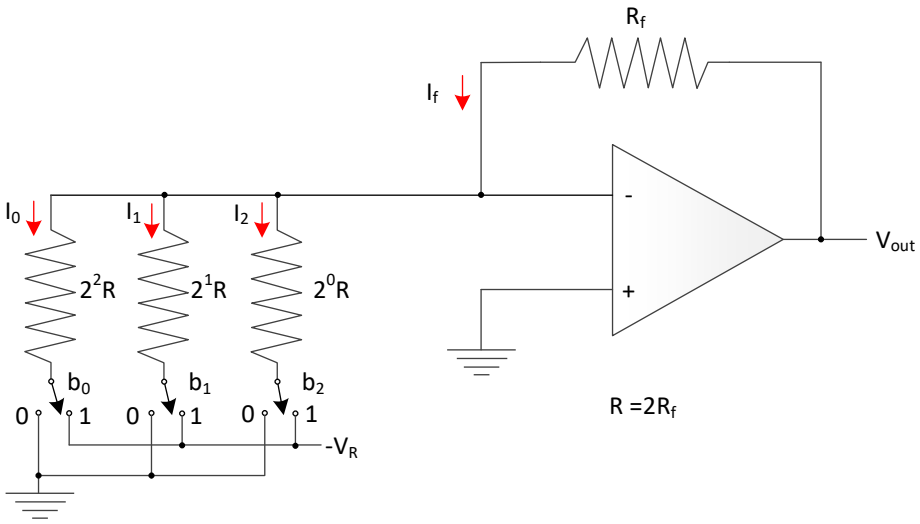


Figure 4. Binary weighted resistors DAC.

First step is to note that the positive input of the amplifier is at the ground, $V_+ = 0$. The input voltage of the amplifier is always $V_{in} = 0$, and thus the voltage at the negative input of the amplifier is also $V_- = 0$. The current flowing through each resistor can be expressed as follows:

$$I_0 = \frac{V_- - b_0 \cdot (-V_R)}{2^2 \cdot R} = \frac{b_0 \cdot V_R}{2^2 \cdot R}$$
$$I_1 = \frac{V_- - b_1 \cdot (-V_R)}{2^1 \cdot R} = \frac{b_1 \cdot V_R}{2^1 \cdot R}$$
$$I_2 = \frac{V_- - b_2 \cdot (-V_R)}{2^0 \cdot R} = \frac{b_2 \cdot V_R}{2^0 \cdot R}$$

To generalize, the current through i-th resistor is:

$$I_i = \frac{b_i \cdot V_R}{2^{N-1-i} \cdot R}$$

The next step is to write equation for the current flowing from the output, I_f (please note that R_f=R/2):

$$I_f = \frac{V_{out} - V_-}{R_f} = \frac{V_{out}}{R_f} = \frac{2 \cdot V_{out}}{R}$$

Finally, the current does not flow into the amplifier, so the current I_f should be equal to the sum of individual currents through the weighted resistors, i.e.:

$$I_f = I_0 + I_1 + I_2$$
$$\frac{2 \cdot V_{out}}{R} = \frac{b_0 \cdot V_R}{2^2 \cdot R} + \frac{b_1 \cdot V_R}{2^1 \cdot R} + \frac{b_2 \cdot V_R}{2^0 \cdot R}$$

$$V_{out} = \left(\frac{b_0}{2^3} + \frac{b_1}{2^2} + \frac{b_2}{2^1}\right) \cdot V_R = \frac{b_0 \cdot 2^0 + b_1 \cdot 2^1 + b_2 \cdot 2^2}{2^3} \cdot V_R = \frac{DAC\_DATA}{2^3} \cdot V_R$$

Example 4-1. Consider a 3-bit DAC discussed above. What is the analog output when $b_2b_1b_0 = 000, b_2b_1b_0 = 001, b_2b_1b_0 = 100, and\ b_2b_1b_0 = 111$. Assume $V_{REF} = 1.5V$.

The transfer function for 3-bit DAC is shown above:

$$V_{out} = \frac{DAC\_DATA}{2^3} \cdot V_R$$

$$b_2b_1b_0 = 000 => V_{out} = 0\,V$$

$$b_2b_1b_0 = 001 => V_{out} = \frac{1}{8}1.5\,V = 0.1875\,V$$

$$b_2b_1b_0 = 100 => V_{out} = \frac{4}{8}1.5\,V = 0.75\,V$$

$$b_2b_1b_0 = 111 => V_{out} = \frac{7}{8}1.5\,V = 1.3125\,V$$

Whereas this implementation is straightforward, the DAC accuracy depends on our ability to make resistors to have resistance as described above: R, 2R, 4R, ..2^{N-1}R. The resistance of the resistors varies widely, especially for larger values of N, which presents a problem in implementation. To remedy this problem an alternative implementation using R-2R binary ladder is used instead.

4.2 R-2R Ladder Network DAC

Figure 5 shows schematic diagram for a 3-bit R-2R ladder network implementation. This ladder arrangement consists of only two resistors, a base resistor with resistance R and a 2R resistor which is twice the value of the base resistor. A pair of R and 2R is used for one input bit as shown in Figure 5. The advantage of this implementation is that it can be fabricated easily as it requires only two values of resistors, the increasing the number of bits does not degrade its performance, and its output resistance remains constant and does not depend on the number of bits.

To analyze the circuit we use Thevenin's theorem and the Superposition theorem. The step-by-step walk through is given in Figure 6, Figure 7, Figure 8, and Figure 9.

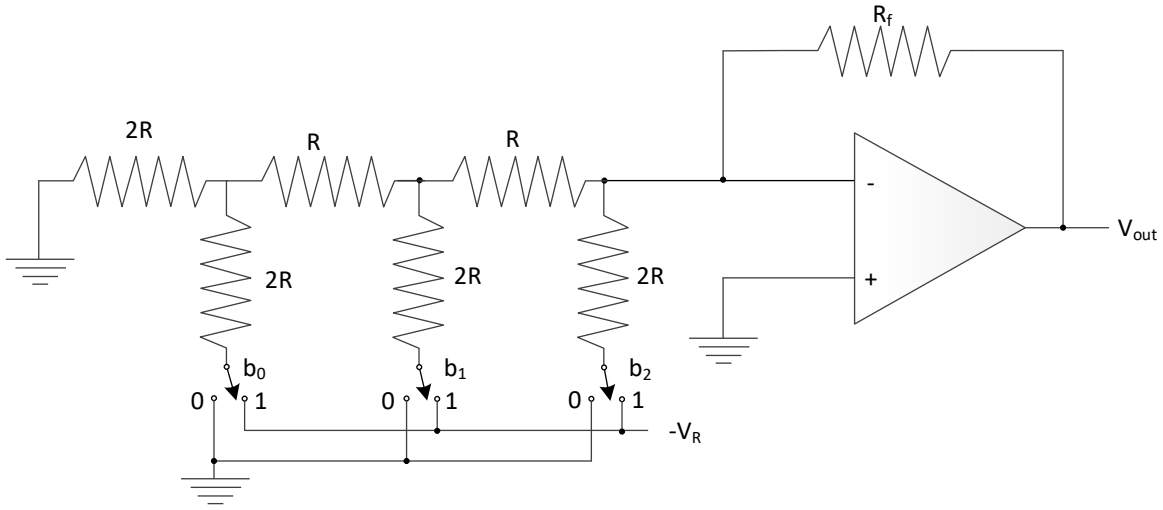
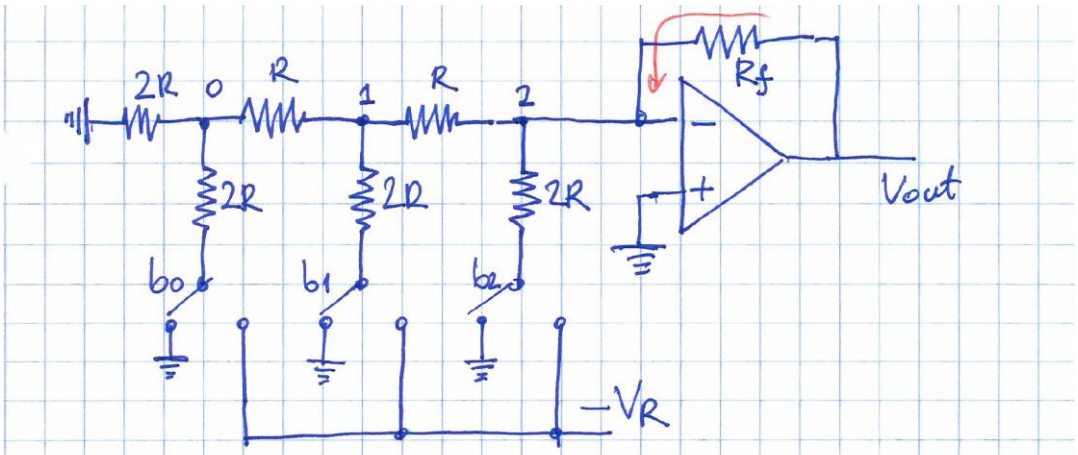
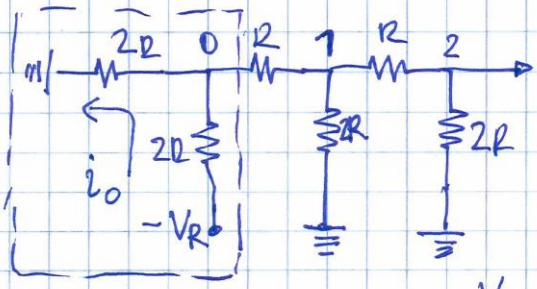


Figure 5. R-2R ladder network DAC.

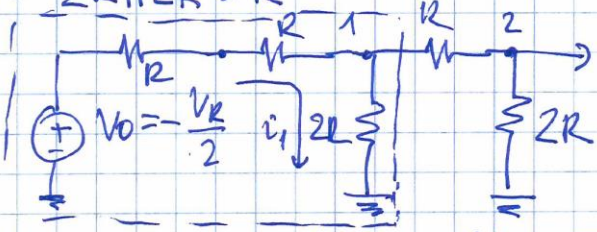


$b_2 b_1 b_0 = 001$



$V_0 = 2R \cdot i_0, \quad i_0 = \frac{-V_R - 0}{2R+2R} = \frac{-V_R}{4R} \Rightarrow V_0 = -\frac{V_R}{2}$

$R_0 = 2R \| 2R = R$



$V_1 = 2R\, i_1, \quad i_1 = \frac{V_0}{2R+2R} = \frac{V_0}{4R} = \frac{-V_R}{8R} \Rightarrow V_1 = \frac{-V_R}{4}$

$R_1 = 2R \| 2R = R$

$V_2 = 2R \cdot i_2, \quad i_2 = \frac{V_1}{4R} = \frac{-V_R}{16R} \Rightarrow V_2 = -\frac{V_R}{8}$

$R_2 = 2R \| 2R = R$

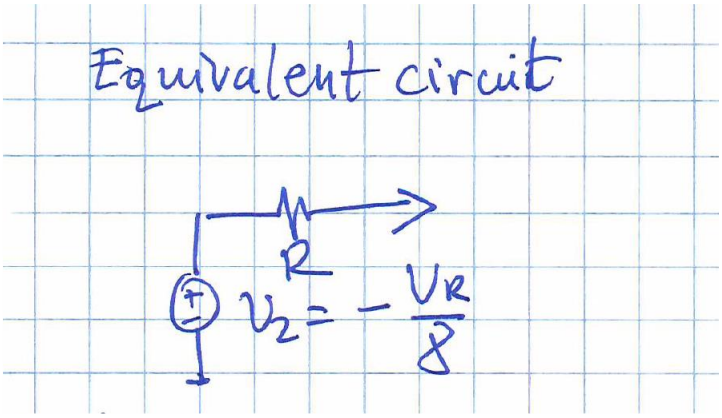


Figure 6. Equivalent circuit for input vector $b_2b_1b_0 = 001$. We mark network nodes near switches 0, 1, and 2, and walk step-by-step creating an equivalent circuit replacing the R-2R ladder. Based on Thevenin's theorem, the equivalent circuit has a power source with voltage corresponding to the potential at given node and equivalent resistance is computed by removing all power sources.

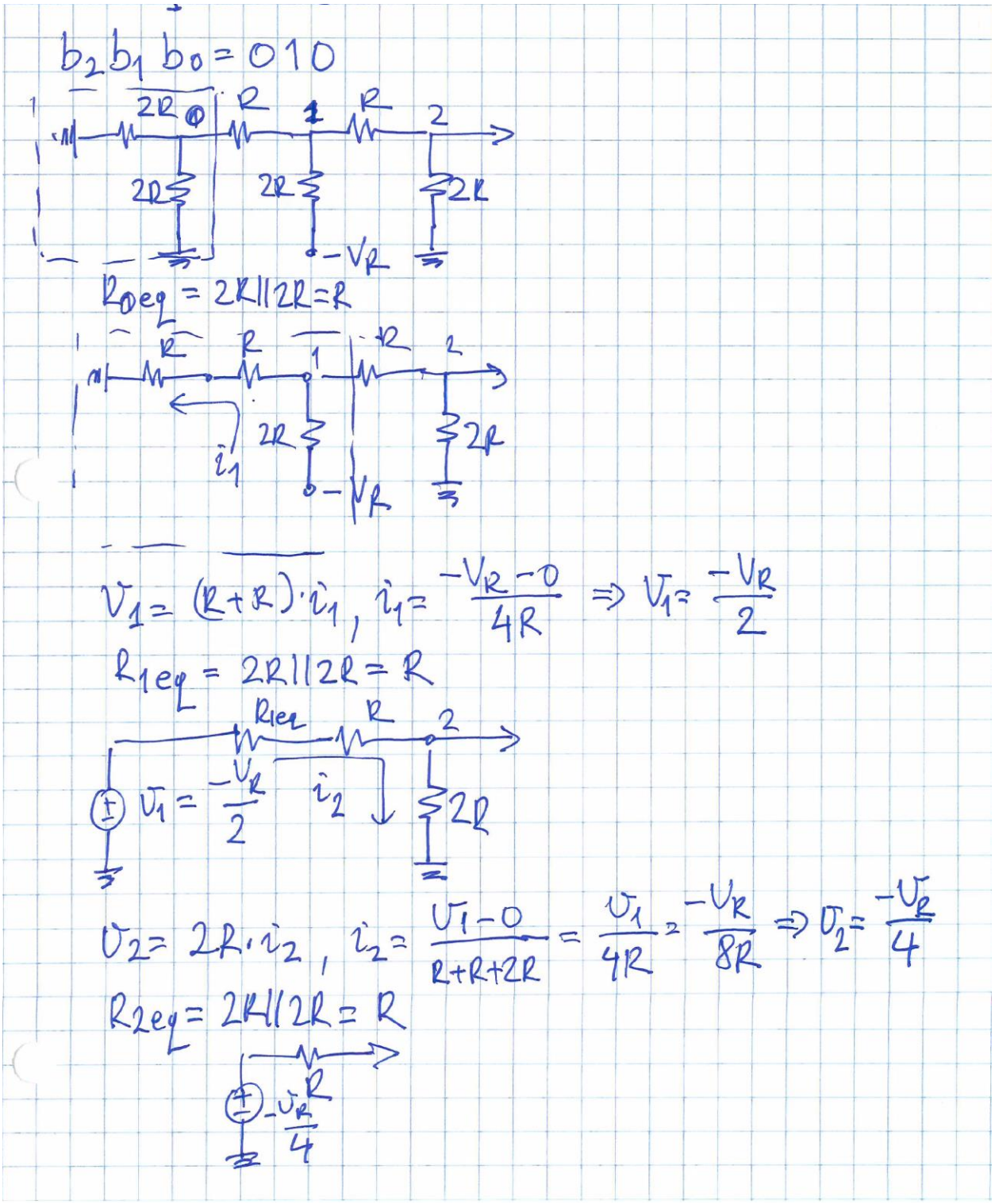


Figure 7. Equivalent circuit for input vector $b_2b_1b_0 = 010$.

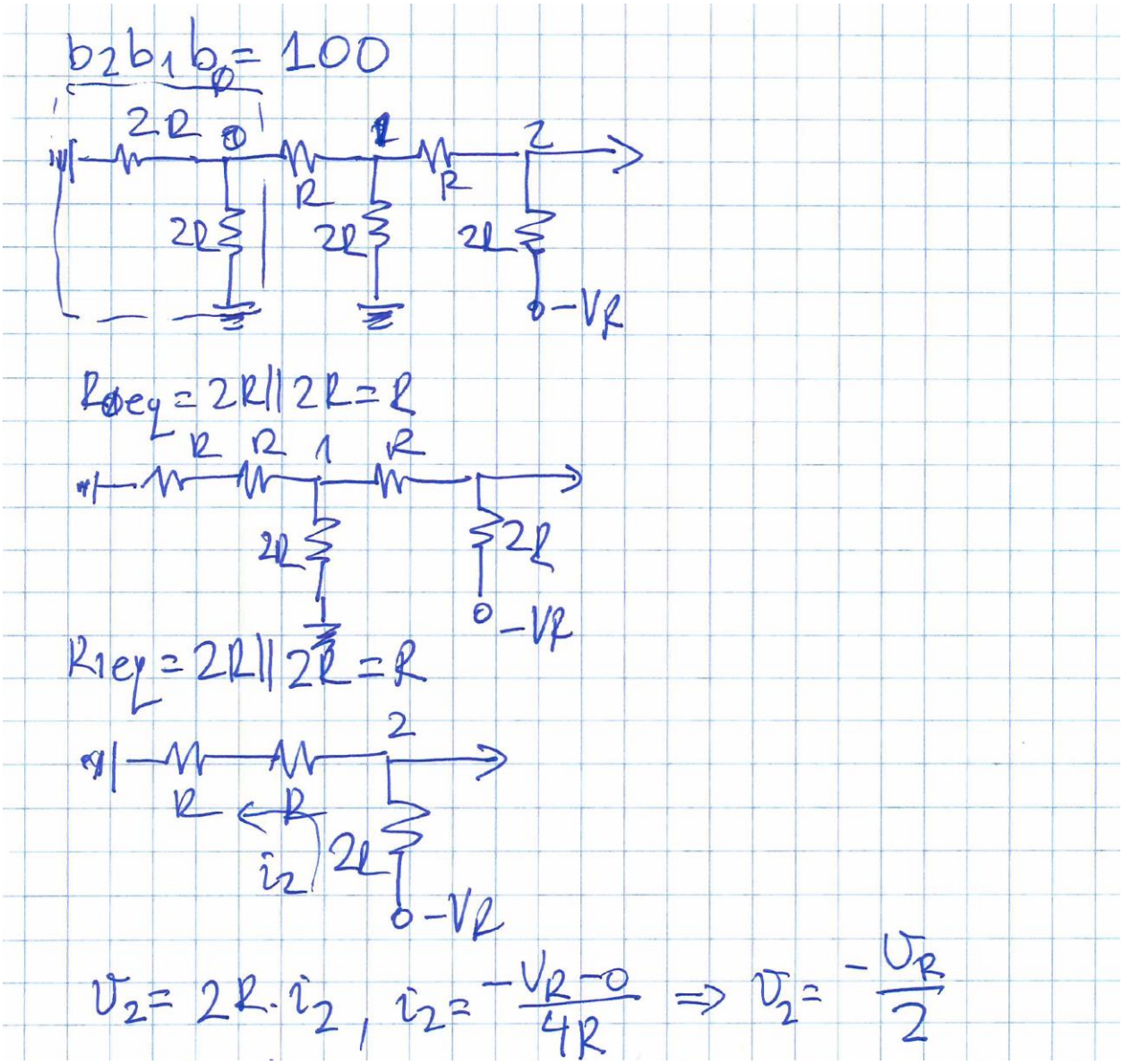


Figure 8. Equivalent circuit for input vector $b_2b_1b_0 = 100$.

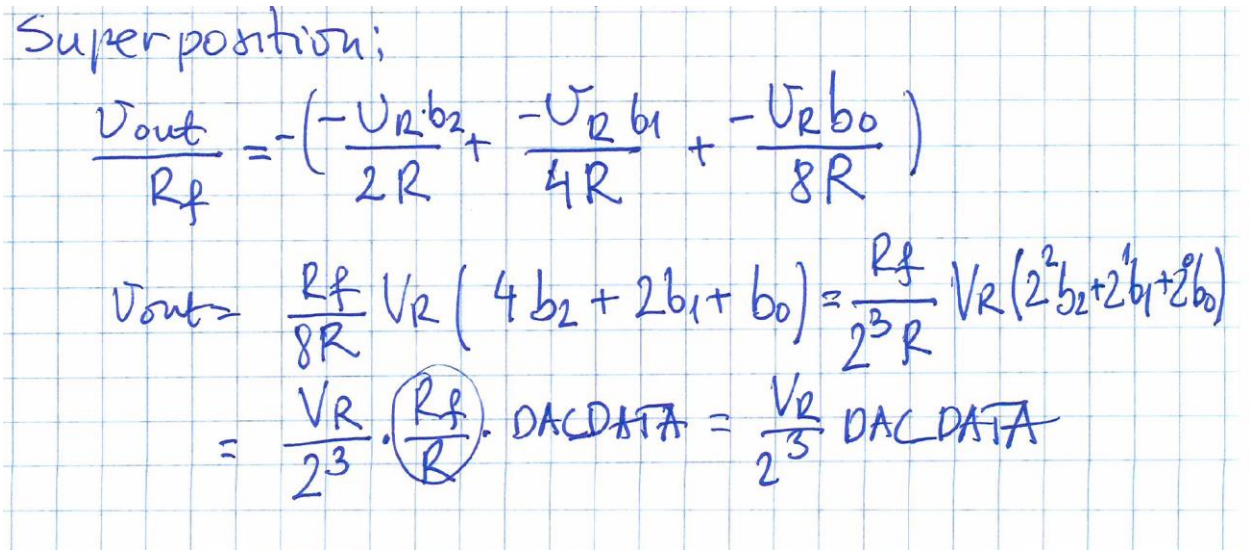


Figure 9. Superposition. Using Superposition theorem and $I_f = I_0 + I_1 + I_2$ we arrive that the desired expression.

5 MSP430's DAC12 Controller

Figure 10 shows a block diagram of MSP430F4618. It includes a 12-channel ADC12, a two-channel DAC12, an analog comparator Comparator_A, and three op amps.

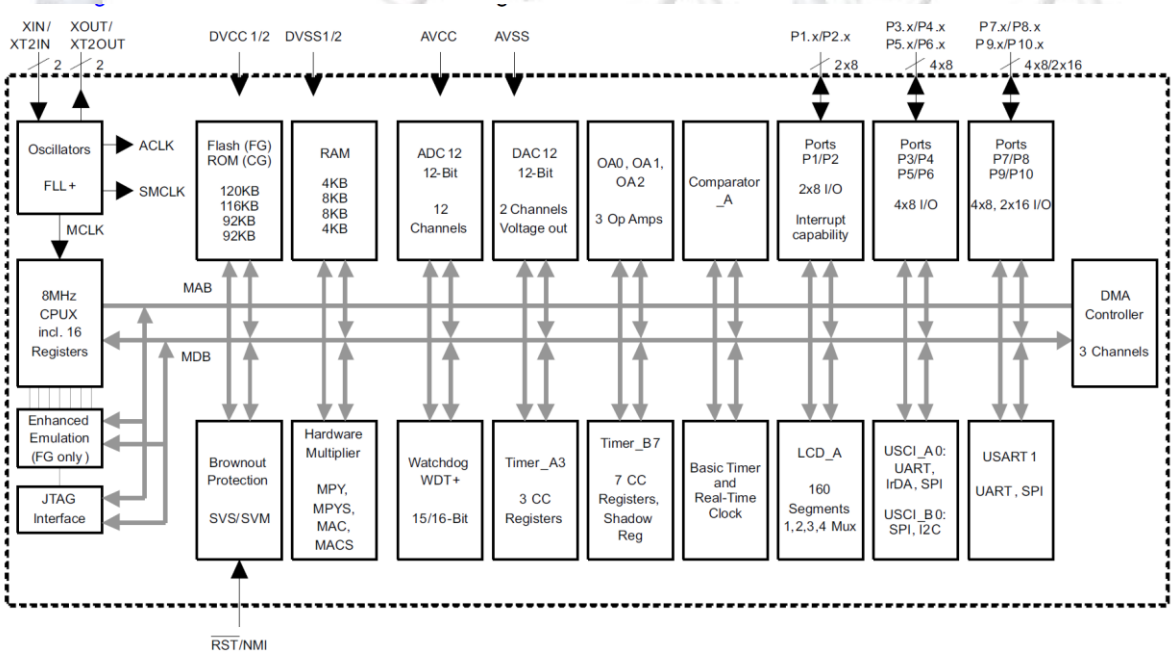


Figure 10. Block diagram of MSP430F4618.

5.1 DAC12 Organization

The MSP430's DAC12 peripheral is a two-channel digital-to-analog converter. Its architecture is based on a ladder of resistors and is conceptually like a potential divider with a large number of settings. Selecting a particular output requires only a particular configuration of switches. The voltage from the potential divider is buffered through an amplifier. The DAC12 can fairly quickly change its output, though typically faster switching requires higher current.

The output voltage of an N-bit DAC is related to its digital input N_{DAC} by the following equation, which is opposite to the equation for the DAC:

$$V_{out} = V_{REF} \cdot \frac{DAC\_DATA}{2^{12}}$$

The input must lie between 0x0000 and 0x0FFF for DAC12 in 12-bit mode. Note: This means that the output voltage cannot quite reach the V_{FS} (4095/4096). The DAC12 can operate in 8-bit mode with inputs between 0x0000 – 0x00FF. There is also a further option for two's complement input, rather than regular unsigned binary; thus, 0x0080 corresponds to output voltage 0 V, and 0x07F corresponds to the maximum output voltage. The DAC12 needs a reference voltage and it usually borrows it from an ADC (typically from the ADC12).

Let us take a look at the DAC12 block diagram shown in Figure 11.

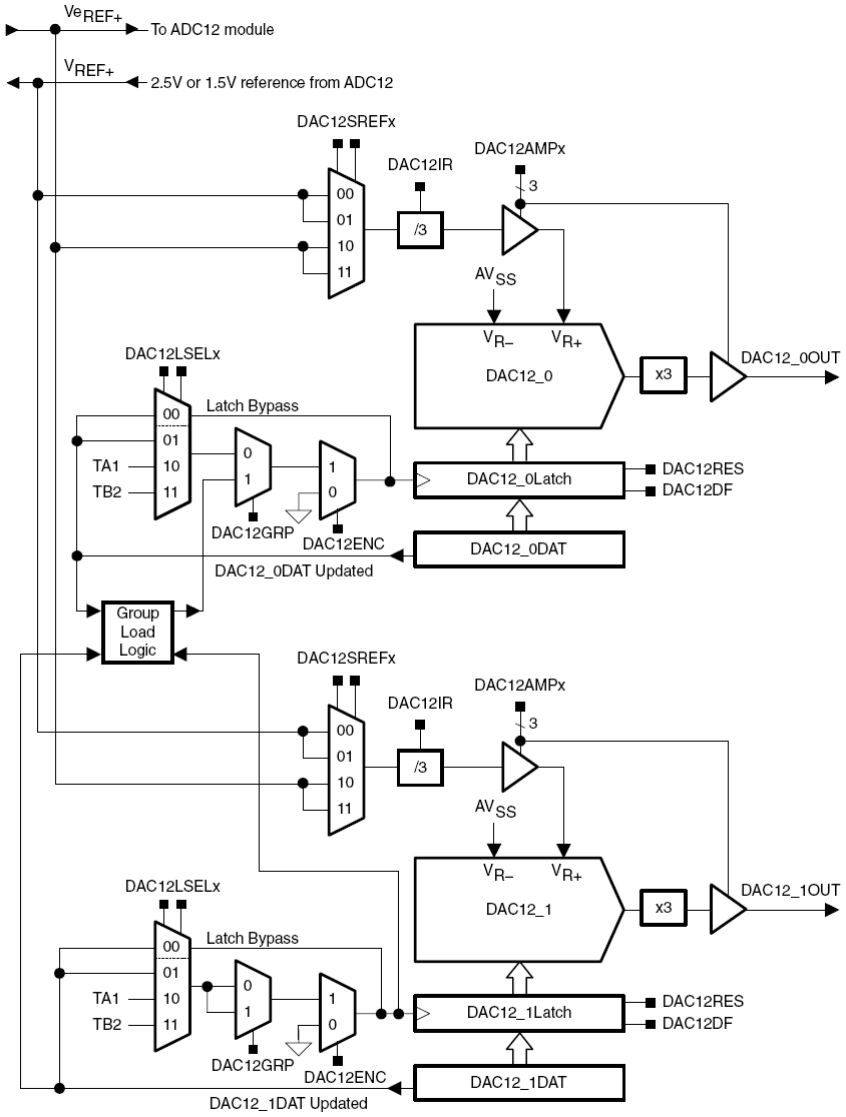


Figure 11. Block diagram of DAC12.

DAC12 Core. The DAC12 can be configured to operate in 8-bit or 12-bit mode using the DAC12RES bit. The full-scale output is programmable to be 1x or 3x the selected reference voltage via the DAC12IR bit. This feature allows the user to control the dynamic range of the DAC12. The DAC12DF bit allows the user to select between straight binary data and 2s compliment data for the DAC. When using straight binary data format, the formula for the output voltage is given below in Figure 12.

Resolution	DAC12RES	DAC12IR	Output Voltage Formula
12 bit	0	0	$Vout = Vref \times 3 \times \frac{DAC12\_xDAT}{4096}$
12 bit	0	1	$Vout = Vref \times \frac{DAC12\_xDAT}{4096}$
8 bit	1	0	$Vout = Vref \times 3 \times \frac{DAC12\_xDAT}{256}$
8 bit	1	1	$Vout = Vref \times \frac{DAC12\_xDAT}{256}$

Figure 12. DAC12 output voltage formula.

DAC12 Outputs. On MSP430FG43x and MSP430FG461x devices, the DAC12 outputs are multiplexed with the port P6 pins and ADC12 analog inputs, and also the V_{eREF+} and P5.1/S0/A12 pins. When DAC12AMPx > 0, the DAC12 function is automatically selected for the pin, regardless of the state of the associated PxSELx and PxDIRx bits. The DAC12OPS bit selects between the P6 pins and the V_{eREF+} and P5.1 pins for the DAC outputs. For example, when DAC12OPS = 0, DAC12_0 outputs on P6.6 and DAC12_1 outputs on P6.7. When DAC12OPS = 1, DAC12_0 outputs on V_{eREF+} and DAC12_1 outputs on P5.1.

Reference Voltage. DAC12 is configured to use either an external reference voltage or the internal 1.5-V/2.5-V reference from the ADC12 module with the DAC12SREFx bits. When DAC12SREFx = {0,1} the VREF+ signal is used as the reference and when DAC12SREFx = {2,3} the VeREF+ signal is used as the reference.

To use an ADC internal reference, it must be enabled and configured via the applicable ADC control bits.

DAC12 Voltage Output. The DAC12_xDAT register can be connected directly to the DAC12 core or double buffered. The trigger for updating the DAC12 voltage output is selected with the DAC12LSELx bits.

When DAC12LSELx = 0 the data latch is transparent and the DAC12_xDAT register is applied directly to the DAC12 core. The DAC12 output updates immediately when new DAC12 data is written to the DAC12_xDAT register, regardless of the state of the DAC12ENC bit.

When DAC12LSELx = 1, DAC12 data is latched and applied to the DAC12 core after new data is written to DAC12_xDAT. When DAC12LSELx = 2 or 3, data is latched on the rising edge from the Timer_A CCR1 output or Timer_B CCR2 output respectively. DAC12ENC must be set to latch the new data when DAC12LSELx > 0.

The DAC12AMPx control bits configure the DAC12's amplifier as shown in Figure 13.

DAC12AMPx	Input Buffer	Output Buffer
000	Off	DAC12 off, output high Z
001	Off	DAC12 off, output 0 V
010	Low speed/current	Low speed/current
011	Low speed/current	Medium speed/current
100	Low speed/current	High speed/current
101	Medium speed/current	Medium speed/current
110	Medium speed/current	High speed/current
111	High speed/current	High speed/current

Figure 13. DAC12 amplifier control.

5.2 DAC12 Control Registers

The DAC12 is a 16-bit peripheral device and two control registers, one for each output channel, DAC12_xCTL. (Figure 14).

DAC12_xCTL, DAC12 Control Register

15	14	13	12	11	10	9	8
DAC12OPS	DAC12SREFx		DAC12RES	DAC12LSELx		DAC12 CALON	DAC12IR
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
DAC12AMPx			DAC12DF	DAC12IE	DAC12IFG	DAC12ENC	DAC12 GRP
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Modifiable only when DAC12ENC = 0

DAC12OPS	Bit 15	DAC12 output select MSP430FG43x and MSP430FG461x Devices: 0 DAC12_0 output on P6.6, DAC12_1 output on P6.7 1 DAC12_0 output on VeREF+, DAC12_1 output on P5.1 MSP430Fx42x0 Devices: 0 DAC12_0 output not available external to the device 1 DAC12_0 output available internally and externally. MSP430FG47x Devices: 0 DAC12_x output not available external to the device 1 DAC12_x output available internally and externally.
DAC12 SREFx	Bits 14-13	DAC12 select reference voltage MSP430FG43x and MSP430FG461x Devices: 00 V_{REF+} 01 V_{REF+} 10 Ve_{REF+} 11 Ve_{REF+} MSP430Fx42x0 and MSP430FG47x Devices: 00 AV_{CC} 01 AV_{CC} 10 V_{REF} (internal from SD16_A or external) 11 V_{REF} (internal from SD16_A or external)
DAC12RES	Bit 12	DAC12 resolution select 0 12-bit resolution 1 8-bit resolution

DAC12 LSELx	Bits 11-10	DAC12 load select. Selects the load trigger for the DAC12 latch. DAC12ENC must be set for the DAC to update, except when DAC12LSELx = 0. 00 DAC12 latch loads when DAC12_xDAT written (DAC12ENC is ignored) 01 DAC12 latch loads when DAC12_xDAT written, or, when grouped, when all DAC12_xDAT registers in the group have been written. 10 Rising edge of Timer_A.OUT1 (TA1) 11 Rising edge of Timer_B.OUT2 (TB2)
DAC12 CALON	Bit 9	DAC12 calibration on. This bit initiates the DAC12 offset calibration sequence and is automatically reset when the calibration completes. 0 Calibration is not active 1 Initiate calibration/calibration in progress
DAC12IR	Bit 8	DAC12 input range. This bit sets the reference input and voltage output range. 0 DAC12 full-scale output = 3x reference voltage 1 DAC12 full-scale output = 1x reference voltage
DAC12 AMPx	Bits 7-5	DAC12 amplifier setting. These bits select settling time vs. current consumption for the DAC12 input and output amplifiers.

DAC12AMPx	Input Buffer	Output Buffer
000	Off	DAC12 off, output high Z
001	Off	DAC12 off, output 0 V
010	Low speed/current	Low speed/current
011	Low speed/current	Medium speed/current
100	Low speed/current	High speed/current
101	Medium speed/current	Medium speed/current
110	Medium speed/current	High speed/current
111	High speed/current	High speed/current

DAC12DF	Bit 4	DAC12 data format 0 Straight binary 1 2s complement
DAC12IE	Bit 3	DAC12 interrupt enable 0 Disabled 1 Enabled
DAC12IFG	Bit 2	DAC12 Interrupt flag 0 No interrupt pending 1 Interrupt pending

Figure 14. Control Register DAC12CTLx.

6 Code Example

Code 1 shows a C program that generates a sinusoidal output using DAC12 with frequency of 10 Hz. We create a lookup table containing 256 samples per one period (T = 0.1 s). The DAC12 is setup as follows: the reference voltage is set to 2.5 V, no amplification is used, and the medium speed/current is used for the output buffer. Please note the the voltage generator is turned on in ADC12 and the time delay is used to wait for the signal to become stable, before using the DAC12 (line 35-37). The main loop of the program is an infinite loop where the processor enters a low-power mode LPM0. The TimerA is set to generate an interrupt 256 times every 0.1 s. The TimerA ISR wakes the CPU and makes sure it remains active after exiting the ISR. In the main loop, the next sample from the lookup table is sent to the DAC12_DAT register and the index is updated before the CPU goes back to LPM0.

Code 2 shows the content of the lookup table created in Matlab.

```
1  /*-------------------------------------------------------------------------
2   * File:        Lab11_D3.c (CPE 325 Lab11 Demo code)
3   * Function:    Sinusoidal wave with DAC (MPS430FG4618)
4   * Description: This C program reconstructs the sinusoidal wave (y=1.25(1+sin(x)))
5   *              from the samples using DAC and outputs at P6.6. WDT is used to
6   *              give an interrupt for every ~0.064ms to wake up the CPU and
7   *              feed the DAC with new value. Connect the oscilloscope to P6.6
8   *              to observe the signal. The interval used to read the samples
9   *              controls the frequency of output signal.
10  * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
11  *              An external watch crystal between XIN & XOUT is required for ACLK
12  *
13  *                      MSP430xG461x
14  *                   -------------------
15  *               /|\|                 XIN|-
16  *                | |                    | 32kHz
17  *                --|RST             XOUT|-
18  *                  |                    |
19  *                  |        DAC0/P6.6|--> sine (10Hz)
20  *                  |                    |
21  * Input:       None
22  * Output:      Sinusoidal wave with 10Hz frequency at P6.6
23  * Author:      Aleksandar Milenkovic, milenkovic@computer.org
24   *-------------------------------------------------------------------------*/
25 #include <msp430fg4618.h>
26 #include "sine_lut_256.h"  /*256 samples are stored in this table */
27
28 void TimerA_setup(void) {
29     TACTL = TASSEL_2 + MC_1;              // SMCLK, up mode
30     TACCR0 = 410;                         // Sets Timer Freq (1048576*0.1sec/256)
31     TACCTL0 = CCIE;                       // CCR0 interrupt enabled
32 }
33
34 void DAC_setup(void) {
35     ADC12CTL0 = REF2_5V + REFON;          // Turn on 2.5V internal ref voltage
36     unsigned int i = 0;
```

```
37      for (i = 50000; i > 0; i--);          // Delay to allow Ref to settle
38      DAC12_0CTL = DAC12IR + DAC12AMP_5 + DAC12ENC;   //Sets DAC12
39  }
40
41  void main(void) {
42      WDTCTL = WDTPW + WDTHOLD;             // Stop WDT
43      TimerA_setup();                       // Set timer to uniformly distribute the
44  samples
45      DAC_setup();                          // Setup DAC
46      unsigned int i = 0;
47      while (1) {
48          __bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled
49          DAC12_0DAT = LUT256[i];
50          i=(i+1)%256;
51      }
52  }
53
54  #pragma vector = TIMERA0_VECTOR
55  __interrupt void TA0_ISR(void) {
56      __bic_SR_register_on_exit(LPM0_bits);  // Exit LPMx, interrupts enabled
57  }
58
```

Code 1. Program for generating a sine wave function $1.25 \cdot (1 + sin(2 \cdot \pi \cdot 10 \cdot t))$.

```
int LUT256[] = { 2048, 2098, 2148, 2198, 2248, 2298, 2348, 2398, 2447, 2496, 2545,
2594, 2642, 2690, 2737, 2784, 2831, 2877, 2923, 2968, 3013, 3057, 3100, 3143, 3185,
3226, 3267, 3307, 3346, 3385, 3423, 3459, 3495, 3530, 3565, 3598, 3630, 3662, 3692,
3722, 3750, 3777, 3804, 3829, 3853, 3876, 3898, 3919, 3939, 3958, 3975, 3992, 4007,
4021, 4034, 4045, 4056, 4065, 4073, 4080, 4085, 4089, 4093, 4094, 4095, 4094, 4093,
4089, 4085, 4080, 4073, 4065, 4056, 4045, 4034, 4021, 4007, 3992, 3975, 3958, 3939,
3919, 3898, 3876, 3853, 3829, 3804, 3777, 3750, 3722, 3692, 3662, 3630, 3598, 3565,
3530, 3495, 3459, 3423, 3385, 3346, 3307, 3267, 3226, 3185, 3143, 3100, 3057, 3013,
2968, 2923, 2877, 2831, 2784, 2737, 2690, 2642, 2594, 2545, 2496, 2447, 2398, 2348,
2298, 2248, 2198, 2148, 2098, 2048, 1997, 1947, 1897, 1847, 1797, 1747, 1697, 1648,
1599, 1550, 1501, 1453, 1405, 1358, 1311, 1264, 1218, 1172, 1127, 1082, 1038, 995,
952, 910, 869, 828, 788, 749, 710, 672, 636, 600, 565, 530, 497, 465, 433, 403, 373,
345, 318, 291, 266, 242, 219, 197, 176, 156, 137, 120, 103, 88, 74, 61, 50, 39, 30,
22, 15, 10, 6, 2, 1, 0, 1, 2, 6, 10, 15, 22, 30, 39, 50, 61, 74, 88, 103, 120, 137,
156, 176, 197, 219, 242, 266, 291, 318, 345, 373, 403, 433, 465, 497, 530, 565, 600,
636, 672, 710, 749, 788, 828, 869, 910, 952, 995, 1038, 1082, 1127, 1172, 1218, 1264,
1311, 1358, 1405, 1453, 1501, 1550, 1599, 1648, 1697, 1747, 1797, 1847, 1897, 1947,
1997, 2047 };
```

Code 2. Lookup table.

7 Exercises