Lab 4 - Data Conversion: Analog-to-Digital and Digital-to-Analog

Alexis Adie and Timonty Lum
ELC 343-L2: Microcomputer Systems

Submitted:
October 25, 2017

Lab 4 - Data Conversion: Analog-to-Digital and Digital-to-Analog

Alexis Adie and Timonty Lum

Department of Electrical and Computer Engineering
The College of New Jersey
2000 Pennington Road, Ewing, NJ 08618, USA
(adiea1, lumt1)@tcnj.edu

I. Introduction

The objective of this lab is to design a PSoC based system to perform real-time digital signal processing. A triangle wave will be generated and a sine wave must be extracted. The range of frequency the system must operate over is 200 Hz to 20 KHz.

The Nyquist Theorem is commonly followed in analog to digital conversions. Essentially, we can digitalize a continuous signal and reconstruct it back as long as the sampling frequency is at least twice the frequency of the input signal.

In theory, reconstruction of discrete time signals into continuous-time signals is often done with a lowpass filter. These lowpass filters are used to remove frequencies twice the highest frequency component. This is also known as impulsive reconstruction. However, a stair-step reconstruction is used in practice, or a zero-order hold function. The zero-order is the method of maintaining each sample value for one sample interval which gives its name the stair-step reconstruction. Some lowpass filtering is used to minimize the edges and generate a quality reconstruction.

II. Procedure

The students began the lab by finding the Fourier series of a triangle wave at 20KHz. They were then tasked to design a lowpass filter that is supposed to remove high-frequency energy and meet the Nyquist criterion. For the DAC, a reconstruction filter was constructed where the input resistance significantly greater than 16k Ohms. The nominal cutoff should be between 20KHz and 50KHz.

The task is to then build a PSoC project that includes SAR ADC and VDAC, with signals routed through analog pins. The ADC is configured with an input voltage range of 0 to 2.048 volts with bypassed internal reference, a conversion rate of at least 600 Ksps, a hardware triggered sample mode, and an internal clock. The VDAC is configured with an output option of high-speed, a data source of CPU, and an external strobe. The clock will be wired at a frequency of 100KHz to the Start of Convert pin and the strobe pin.

Observe and record outputs with the following inputs of the triangle wave: 200Hz, 2000Hz, and 20,000Hz. Pass through algorithms that will convert the triangle wave to one the is more sinusoidal.

III. Design

The SAR ADC and the VDAC were built in the schematics of PSoC and routed to the appropriate pins. The filter was added into the design to complete the system.

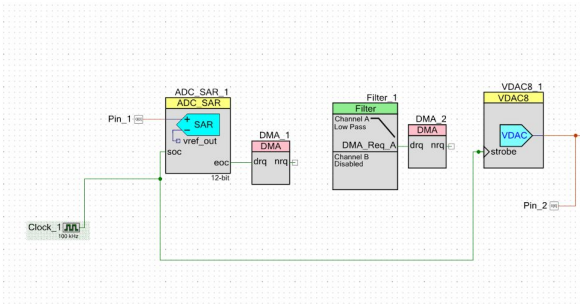

Figure 3.1: PSoC Schematic

The team was tasked with creating a low-pass filter with a cutoff frequency of 20 kHz.
The value of the capacitor was chosen to be $.15\mu F$ and with equation 1, the resistance was calculated.

(1) $R = \frac{1}{2\pi F \varepsilon}$

where R=resistance, F=cutoff frequency,
ε = capacitance
The group calculated the resistance to be 53 Ω.
With all of the values found, a low-pass filter was designed as seen in Figure 2.1.

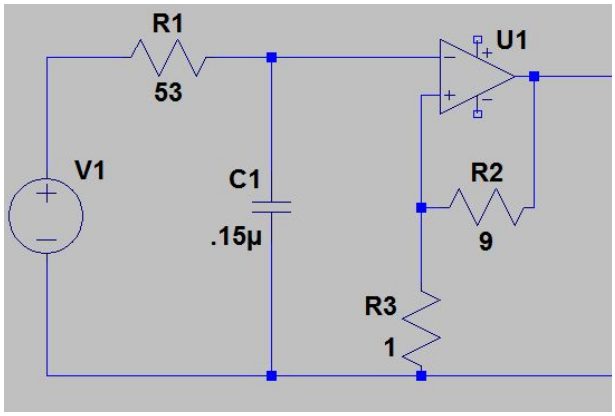

Figure 3.2: Low-Pass Filter Design

$$f(x) = \frac{8}{\pi^2} \sum_{n=1,3,5,\ldots}^{\infty} \frac{(-1)^{(n-1)/2}}{n^2} \sin\left(\frac{n\pi x}{L}\right).$$

Figure 3.3: Fourier Series of a Triangle Wave

Figure 3.3 shows the Fourier Series of a triangle wave. Having a Fourier Series representation allows easier analyzation of the system.

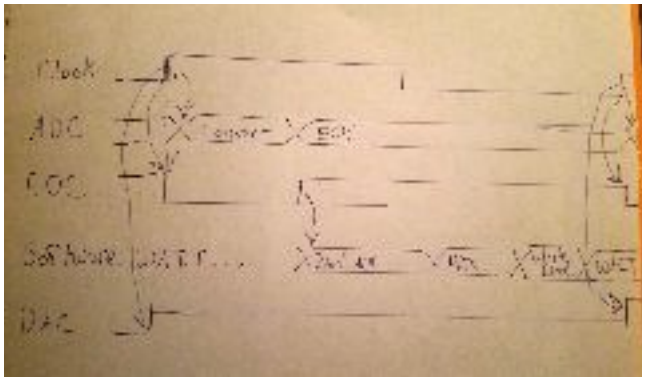

Figure 3.4: Timing Diagram

Figure 3.4 shows a timing diagram explaining when each part of the project begins operation.

IV. Result

A. *Oscilloscope Readings*

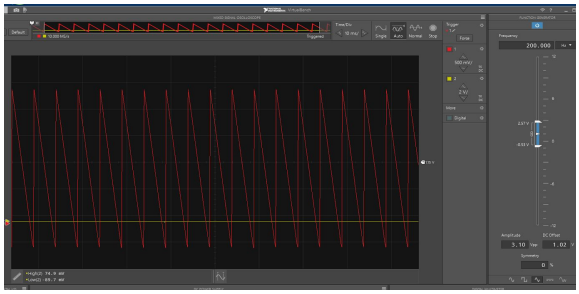

Figure 4.1: Triangle Wave Input at 200 Hz

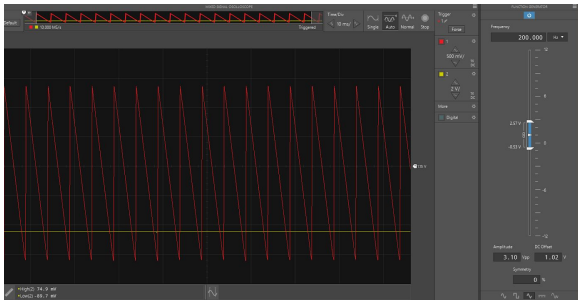

Figure 4.2: Triangle Wave Input at 2,000 Hz

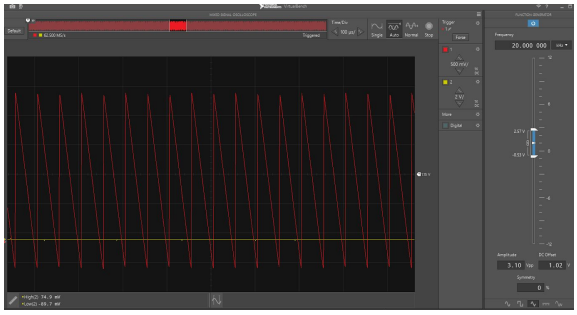

Figure 4.3: Triangle Wave Input at 20,000 Hz

Figure 4.4: DAC Output Wave at 200 Hz

Figure 4.5: DAC Output Wave at 2,000 Hz

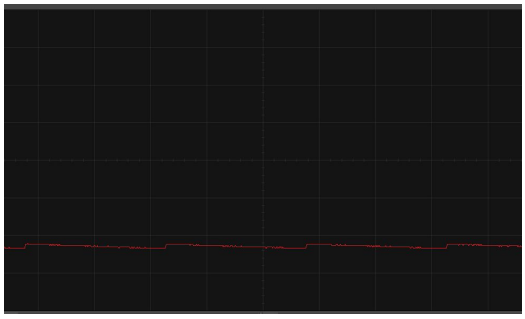

Figure 4.6: DAC Output Wave at 20,000 Hz

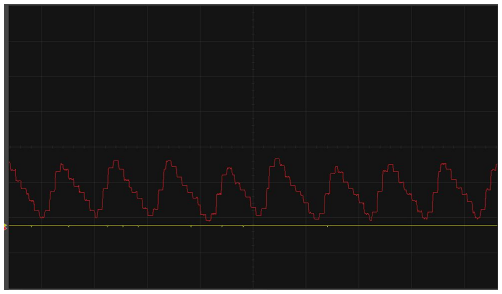

Figure 4.7: LPF Output Wave at 200 Hz

Figure 4.8: LPF Output Wave at 2,000 Hz

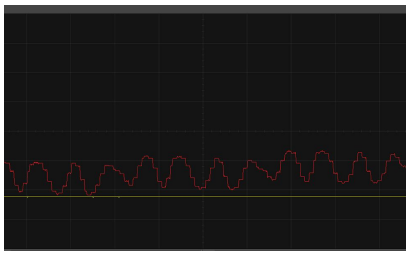

Figure 4.9: LPF Output Wave at 20,000 Hz

B. Debug Experiences

While the team was able to achieve a sinusoidal-like output, it was unable to produce a filter which depleted the "stairstep" reconstruction errors made by the DAC. Another failure was the sine lookup table algorithm which did not seem to make the function more sinusoidal.

IV. Conclusion

The objective was to design a PSoC based system to perform real-time digital signal. A first order low pass filter was designed with a cutoff at 20KGz. A

reconstructor filter is then designed to reduce the the stairstep reconstruction errors. These were incorporated into the Analog to Digital Convert (ADC) and Digital to Analog Converter (DAC). After routing the signals through pins and configuring the clock signal, the algorithm was passed through the system so transform the triangle wave to be more sinusoidal. The output was then captured when the input values were changed to 200Hz, 2000Hz, and 20,000Hz. The triangle wave does become more sinusoidal but the edges were not fully removed. So the objective was successfully completed only partially.

V. Code

```
#include <project.h>

/* Defines for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE
(CYDEV_PERIPH_BASE)
#define DMA_1_DST_BASE
(CYDEV_PERIPH_BASE)


/* Defines for DMA_2 */
#define DMA_2_BYTES_PER_BURST 1
#define DMA_2_REQUEST_PER_BURST 1
#define DMA_2_SRC_BASE
(CYDEV_PERIPH_BASE)
#define DMA_2_DST_BASE
(CYDEV_PERIPH_BASE)

int main()
{

/* Variable declarations for DMA_1
and DMA_2*/

uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];
uint8 DMA_2_Chan;
uint8 DMA_2_TD[1];

/* DMA Configuration for DMA_1 */
DMA_1_Chan =
DMA_1_DmaInitialize(DMA_1_BYTES_PER_
BURST, DMA_1_REQUEST_PER_BURST,

HI16(DMA_1_SRC_BASE),
HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0],
1, DMA_1_TD[0], 0);
CyDmaTdSetAddress(DMA_1_TD[0],
LO16((uint32)ADC_SAR_1_SAR_WRK0_PTR)
,
LO16((uint32)Filter_1_STAGEA_PTR));
CyDmaChSetInitialTd(DMA_1_Chan,
DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);

/* Variable declarations for DMA_2
*/
/* Move these variable declarations
to the top of the function */

/* DMA Configuration for DMA_2 */
DMA_2_Chan =
DMA_2_DmaInitialize(DMA_2_BYTES_PER_
BURST, DMA_2_REQUEST_PER_BURST,

HI16(DMA_2_SRC_BASE),
HI16(DMA_2_DST_BASE));
DMA_2_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_2_TD[0],
1, DMA_2_TD[0], 0);
CyDmaTdSetAddress(DMA_2_TD[0],
LO16((uint32)Filter_1_HOLDA_PTR),
LO16((uint32)VDAC8_1_Data_PTR));
CyDmaChSetInitialTd(DMA_2_Chan,
DMA_2_TD[0]);
```

```
CyDmaChEnable(DMA_2_Chan, 1);

 /* Start the components */
Clock_1_Start();
VDAC8_1_Start();
Filter_1_Start();
ADC_SAR_1_Start();
Filter_1_SetCoherency(Filter_1_CHANN
EL_A, Filter_1_KEY_LOW);

//sine lookup table
int sineTable[1025];

int i, j, k;
  const int magic[33] = {
      0x691864, 0x622299, 0x2CB61A,
0x461622,
      0x62165A, 0x85965A, 0x0D3459,
0x65B10C,
      0x50B2D2, 0x4622D9, 0x88C45B,
0x461828,
      0x6616CC, 0x6CC2DA, 0x512543,
0x65B69A,
      0x6D98CC, 0x4DB50B, 0x86350C,
0x7136A2,
      0x6A974B, 0x6D531B, 0x70D514,
0x4EA714,
      0x5156A4, 0x393A9D, 0x714A6C,
0x755555,
      0x5246EB, 0x916556, 0x7245CD,
0xB4F3CE,
      0x6DBC7A
  };
  k = 0;
  for (i = 0; i < 33; i++) {
      for (j = 0; j < 24; j += 3) {
      sineTable[k++] = ((magic[i] >>
j) & 7) - 4;
      }
  }
  sineTable[1] = 51472;

//for loop which estimates the value
of the output value
  for (i = 3; i > 0; i--) {
      for (j = i; j <= 256; j++) {
      k = sineTable[j - 1];
      sineTable[j] = k +
sineTable[j];
      sineTable[513-j] = k;
      sineTable[511+j] = -k;
      sineTable[1025-j] = -k;
      }
  }
  sineTable[768] = -0x800000;
//Writes values to output
for (i = 0; i < 1025; i++) {
Pin_2_Write(sineTable[i]);
   }
}
```