Testing Arduino Due Sampling Rate

Cole Sheyka

April 2023

1 Abstract

This experiment aimed to test the limits of the Arduino Due board in outputting a square wave with varying frequency solver methods, and extra code. The purpose of these tests is to discover what the actual performance of the hardware is.

2 Introduction

The goal of these tests is to find the maximum frequency an Arduino Due can output a square wave using code compiled with Simulink. Adding more code to a Simulink model which needs to be uploaded to an Arduino Due may potentially affect the microcontroller's performance in performing multiple tasks at the same time.

The SAM3X8E is a microcontroller unit (MCU) that is used in the Arduino Due board. It is based on the ARM Cortex-M3 architecture and has a clock speed of 84 MHz. The MCU has a 32-bit data bus and can access up to 512 KB of flash memory and 100 KB of SRAM.

When a Simulink model is compiled for the Arduino Due board, the generated code is executed by the SAM3X8E MCU. Adding more code to the model will increase the amount of processing time required by the MCU to execute the code. This increased processing time can potentially limit the maximum frequency at which the board can output a square wave.

3 Simulink Set Up

Figure 1 shows the Simulink block diagrams for tests 1-4 with the exception of the second order transfer function block shown at the top of the figure. The first portion of Test 2 involved using a first order transfer function block which is not shown above. Test 1 was run with only the blocks in its respective rectangle. In tests 2 and 3, the square wave was still observed through pin 9, but the extra

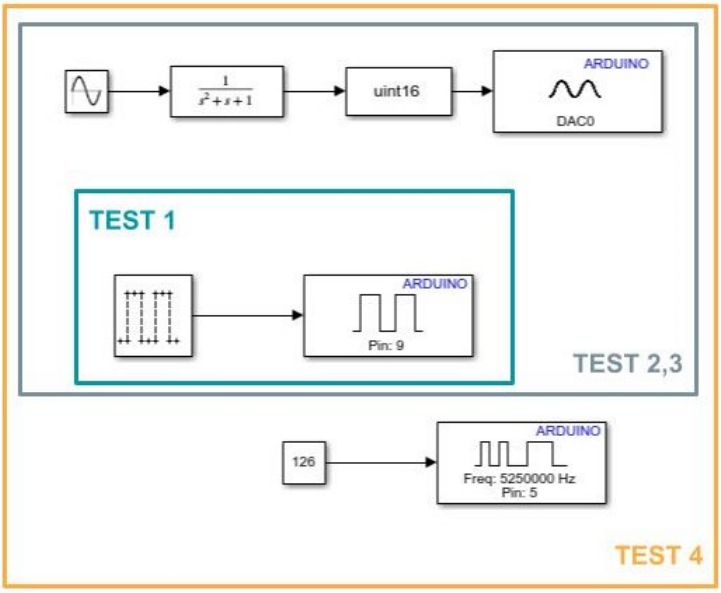

Figure 1: Simulink program showing the blocks used to run 4 tests

blocks are there to try and hinder the Due's performance. In Test 4, all of the blocks were still present but instead of viewing a square wave out of pin 9 with the pulse generator, a pulse width modulation (PWM) block was used to view the signal out of pin 5.

Before selecting and adjusting the blocks seen in Figure 1, the model was configured in hardware settings. For tests 1,2 and 4, the solver was left to 'auto' which was by default set to ode3. Two different solvers, ode1 and ode1be, were used in test 3 to see how different solvers can affect performance. Single simulation output was unselected. Overrun protection was also enabled and was by default configured to pin 13 on the Due. By checking this box, pin 13 on the Arduino is set to go HIGH if the MCU is not able to complete all of its tasks in the specified sampling period. When pin 13 is HIGH on the Due, the L light turns on which is a solid indicator that the square wave being output is not accurate.

4 Test 1: Basic Pulse Generator

For Test 1, a pulse generator block was used to code a square wave with 50% duty cycle to be output from pin 9 on the Due. Specifically, the block parameters were set such that the period of the pulse was 10 samples and the pulse width was 5 samples. By changing the pulse type to 'sample based,' the user is able to specify the frequency the output signal should be. The time-based period of the output signal can be backed out by multiplying the sample period of the pulse with the sampling frequency.

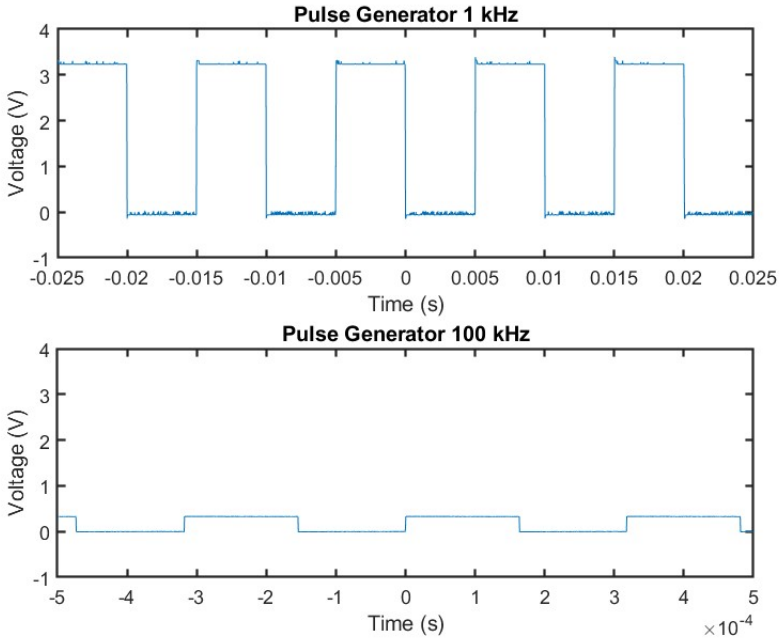

Figure 2: Simulink program showing the blocks used to run 4 tests

5 Test 2a:

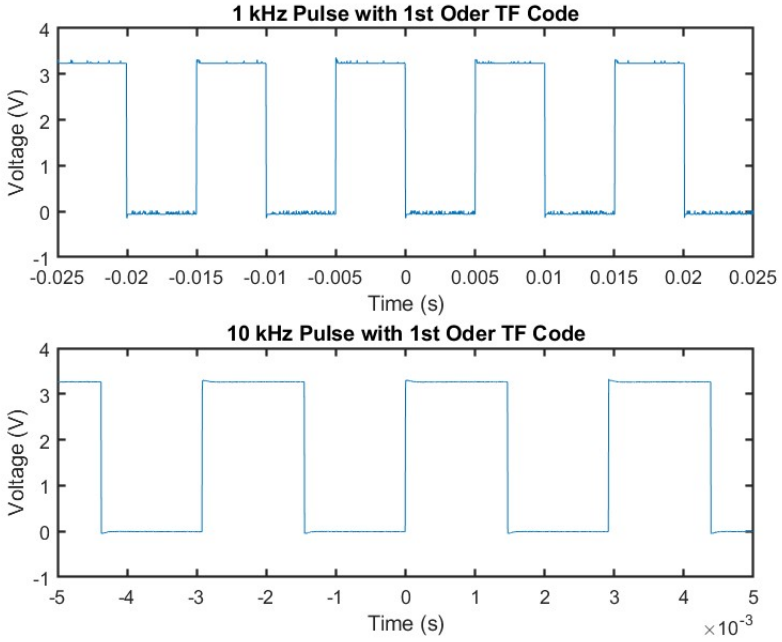

Figure 3: Simulink program showing the blocks used to run 4 tests

The purpose of this test was to test the MCU's performance as more code is added into the Simulink block diagram. For the first part of this test, a first order transfer function block was placed, taking a

sine wave as an input. The output of the transfer function block fed into a digital-to-analog block for the Arduino, however the data seen in Figure 3 was saved from the output of pin 9 as mentioned before.

6 Test 2b: Pulse Generator with Extra Code

For the second part of this test, the first order transfer function block was exchanged for a second order transfer function. The purpose of this was to add more complexity and steps for the MCU to process in the given sample times.

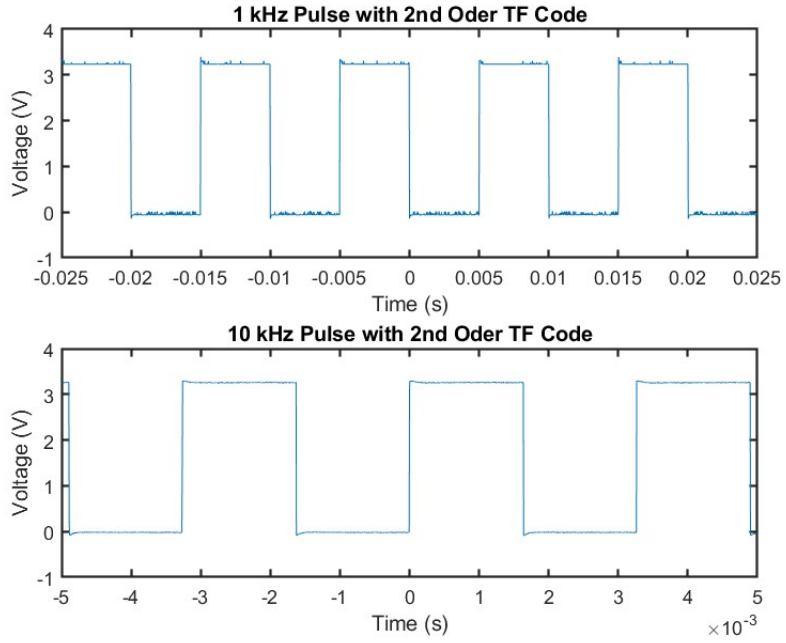

Figure 4: Simulink program showing the blocks used to run 4 tests

7 TEST 3a: Using the Euler Method

For the first portion of this test, the solver, ode3, was replaced by ode1, which is useful for solving simple first-order differential equations. The ode1 solver works by using the Euler method, which approximates the solution by using the derivative at the current time to predict the value of the function at the next time step. This process is repeated over the simulation time to approximate the solution.

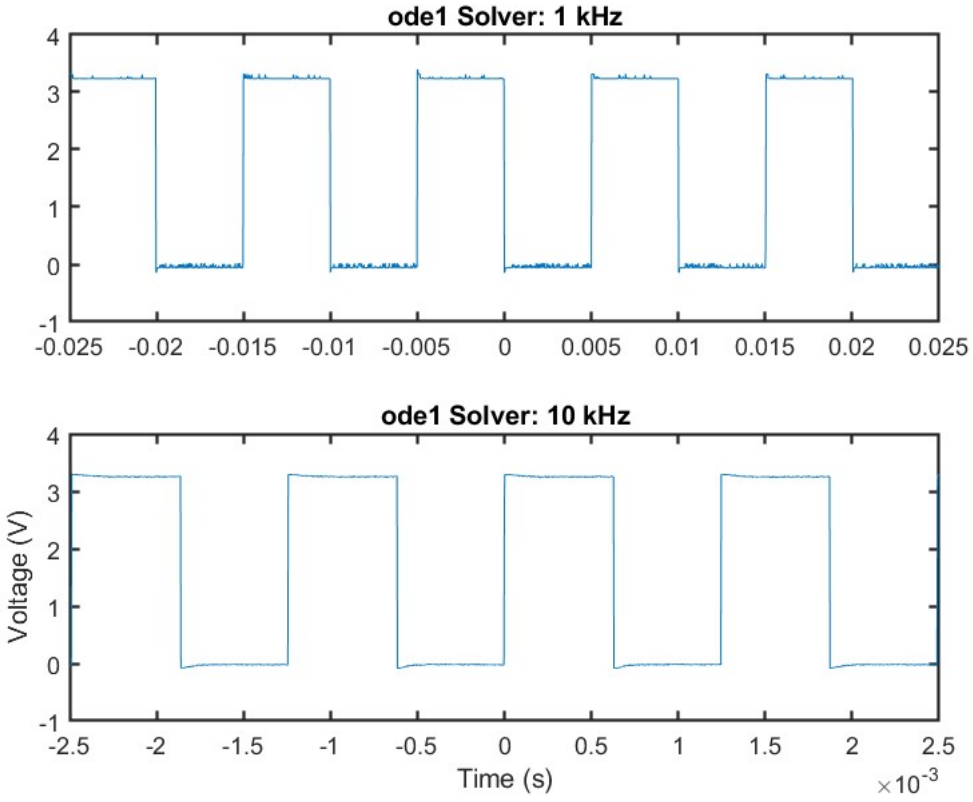

Figure 5: Simulink program showing the blocks used to run 4 tests

8 Test 3b: Using the Backwards Euler Method

The second solver used in this test was ode1be, which is the backward Euler method. Ode1be involves taking a single backward step, using the derivative at the next time step to predict the value of the function at the current time step. The backward Euler method has better stability properties, making it more accurate and reliable than the Euler method. However, ode1be is more computationally expensive and it is apparent that the MCU could not produce an accurate signal as efficient as the other solvers could.

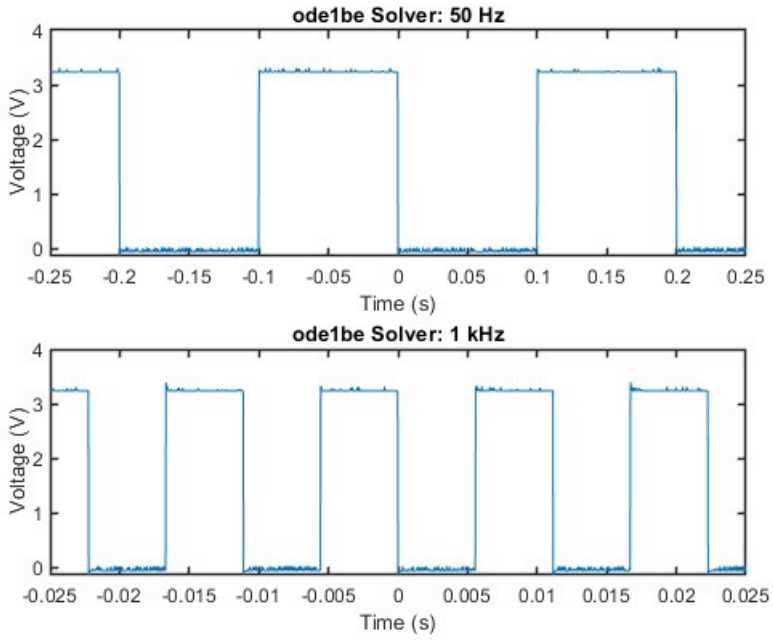

Figure 6: Simulink program showing the blocks used to run 4 tests

9 Test 4: PWM Output Signal with Extra Code

In the final test, the solver was switched back to ode3 and the signal was observed through pin 5 on the Due. The fastest frequency the user can set the PWM block to is 21 MHz. As seen in Figure 7, at 14 MHz, the settling time in the overshoot and undershoot of the signal was noticeably large. The 21 MHz output signal did not resemble a square wave at all, so it was not included in the plots.

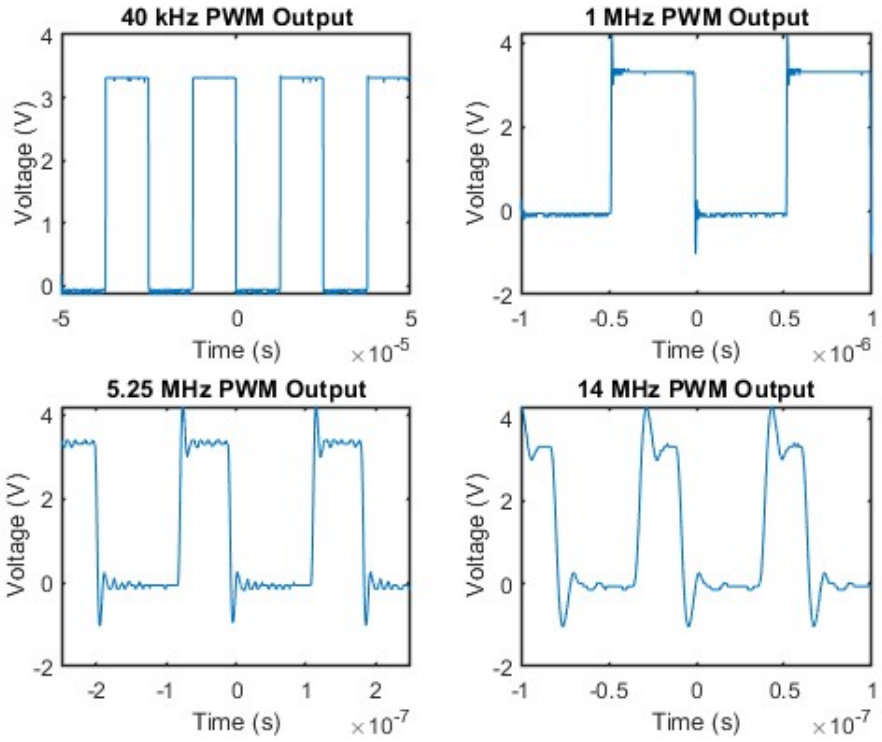

Figure 7: Simulink program showing the blocks used to run 4 tests

10 Coding to the Metal

The SAM3X8E MCU has a variety of peripherals and features that can be utilized to improve the performance of the board. For example, the MCU has a high-speed direct memory access (DMA) controller that can be used to transfer data between peripherals and memory without CPU intervention. This can help to reduce the CPU load and improve the overall performance of the board.

In addition, the MCU has a variety of clock sources and power management features that can be used to optimize the performance and power consumption of the board. For example, the MCU has a programmable PLL (phase-locked loop) that can be used to generate high-speed clocks from a lower-speed source.

11 Conclusion

While adding more code to a Simulink model can potentially affect the performance of the microcontroller and limit the maximum frequency at which the board can output a square wave, it is possible to optimize the code and utilize the MCU's peripherals and features to improve the board's performance. By testing the maximum frequency at which the board can produce a square wave in Simulink, you can

optimize the code and leverage the MCU's features to maximize the board's potential. Ultimately, understanding the capabilities and limitations of the SAM3X8E MCU is crucial for improving performance of the Due.

12 MATLAB Code

```
clc, clear, close all;

%%-------------------------------------
% Project 3: Testing DUE Specs
% Written by Cole Sheyka 4/17/23
%%-------------------------------------

%% IMPORT DATA FROM 4 TESTS
% TEST 1: Using pulse generator to output a square wave
test1_5 = readmatrix("scope_1.csv");
test1_1k = readmatrix("1T1k.csv");

% TEST 2: Adding more code
% First order TF
test2_5_1 = readmatrix("scope_2.csv");
test2_1k_1 = readmatrix("2T1k.1o.csv");

% Second order TF
test2_5_2 = readmatrix("scope_3.csv");
test2_1k_2 = readmatrix("2T1k.2o.csv");

% TEST 3: Adjusting solver settings
% Kept 2nd order TF in Simulink code
% Using euler (ode1)
test3_5_1 = readmatrix("scope_4.csv");
test3_1k_1 = readmatrix("3T1k.1.csv");

% Using backwards Euler (ode1be)
test3_5_1be = readmatrix("3T5.1be.csv");
test3_1k_1be = readmatrix("3T1k.1be.csv");
```

```
% TEST 4: Testing PWM
% Kept 2nd order TF in Simulink code
test4_40k = readmatrix("4f40.csv");
test4_1M = readmatrix("4f1M.csv");
test4_525M = readmatrix("4f5.25M.csv"); % 5.25 MHz PWM
test4_14M = readmatrix("4f14M.csv");

% Making matrix with all of the data
M = [test1_1k test1_5 ...
    test2_1k_1 test2_5_1 test2_1k_2 test2_5_2 ...
    test3_1k_1 test3_5_1 test3_1k_1be test3_5_1be ...
    test4_40k test4_1M test4_525M test4_14M];


%% BUILD PLOTS
% TEST 1: Using pulse generator to output a square wave
figure;
subplot(2,1,1)
plot(M(:,1),M(:,2))
title("Pulse Generator 1 kHz")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)

subplot(2,1,2)
plot(M(:,3),M(:,4))
title("Pulse Generator 100 kHz")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)
ylim([-1,4])

% TEST 2: Adding more code
```

```
% First order TF
figure;
subplot(2,1,1)
plot(M(:,5),M(:,6))
title("1 kHz Pulse with 1st Oder TF Code")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)

subplot(2,1,2)
plot(M(:,7),M(:,8))
title("10 kHz Pulse with 1st Oder TF Code")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)
ylim([-1,4])

% Second order TF
figure;
subplot(2,1,1)
plot(M(:,9),M(:,10))
title("1 kHz Pulse with 2nd Oder TF Code")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)

subplot(2,1,2)
plot(M(:,11),M(:,12))
title("10 kHz Pulse with 2nd Oder TF Code")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)
ylim([-1,4])
```

```
% TEST 3: Adjusting solver settings
% Kept 2nd order TF in Simulink code
% Using euler (ode1)
figure;
subplot(2,1,1)
plot(M(:,13),M(:,14))
title("ode1 Solver: 1 kHz")
set(gca,'linewidth',1.15)

subplot(2,1,2)
plot(M(:,15),M(:,16))
title("ode1 Solver: 10 kHz")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)
ylim([-1,4])

% Using backwards Euler (ode1be)
figure;
subplot(2,1,2)
plot(M(:,17),M(:,18))
title("ode1be Solver: 1 kHz")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)

subplot(2,1,1)
plot(M(:,19),M(:,20))
title("ode1be Solver: 50 Hz")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)
```

```
ylim([-1,4])

% TEST 4: Testing PWM
% Kept 2nd order TF in Simulink code
figure;
subplot(2,2,1)
plot(M(:,21),M(:,22)) % 40kHz
title("40 kHz PWM Output")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)

subplot(2,2,2)
plot(M(:,23),M(:,24)) % 1MHz
title("1 MHz PWM Output")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)

subplot(2,2,3)
plot(M(:,25),M(:,26)) % 5.25MHz
title("5.25 MHz PWM Output")
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)

subplot(2,2,4)
plot(M(:,27),M(:,28)) % 14MHz
title("14 MHz PWM Output")
xlim([-1e-7,1e-7])
xlabel('Time (s)')
ylabel('Voltage (V)')
set(gca,'linewidth',1.15)
```