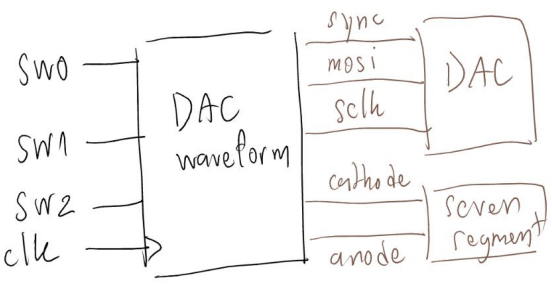
Introduction

This lab involves a relatively new-introduced peripheral, digital-to-analog converter (DAC), which changes encoded signals to specific voltage. Furthermore, an imperfect of button, debouncing, is also studied via a new implementation to control the block by using buttons shown to VGA display. Also, a test bench, an option to debug and check the correctness of a module, is also used in this lab. Through this lab, many lessons are learned to cope with usual problem from expected trouble from peripherals and regular components. We can finally accomplish last result and solve problems occuring between the process.

Discussion

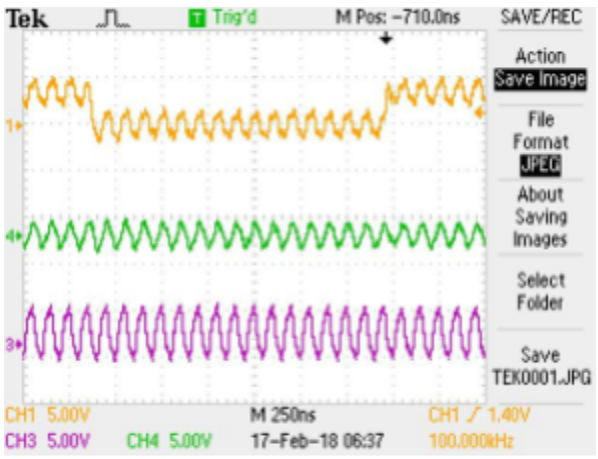
Part 1: DAC - Waveform Generation



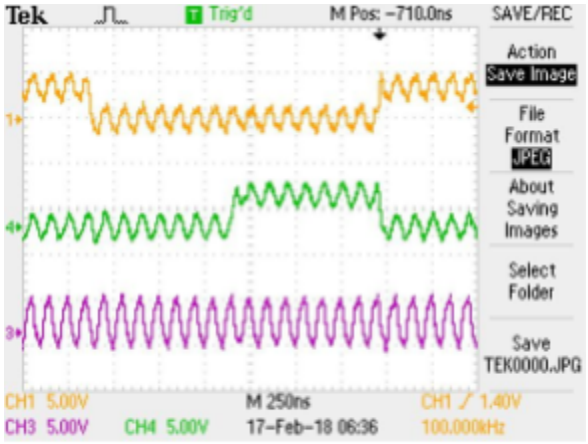
In this section, the module tried to create digital signals in the digital-to-analog converter (DAC) form whose signals consists of `sync`, `data`, and `sclk`; `sync` signals the DAC that the data (through `data`) is about to be sent at rising edges of `sclk`.

The block diagram roughly looks like the picture on the left side. The inputs is for selecting what kind of data to generate; in this module, the names indicates them: constant voltage, sawtooth waveform, and triangle waveform. Other than the inputs, the module handles itself. According to the datasheet, there are three signals that the module have to arrange: `sync`, `mosi`, `sclk`. Because it is required that DAC updates every 10 ms (100kHz), the clock divider is used. `sync` is brought low when the clock enable goes high (every 10 ms) and is brought high when all data (16 bits) are sent out, which are kept by `shift_state`. The data is sent via `mosi` (master-out-slave-in). Because the module keeps data bits in left shift register and assumes that the MSB is the first bit, `mosi` is always at the MSB of the shift register. Also, the bits out is not only the voltage digital value but also a configuration of DAC; in this module, to keep it simple, the default configuration, all bits are 0, is used. As mentioned, there are three output waveforms, so the values assigned to the shift register are stored separately and assigned according to the input switches. The constant voltage is assigned to the maximum voltage. Because it is required that the frequency of sawtooth waveform is 3.125kHz that is equivalent to 32 steps for each 10-ms update, so the digital value is set from 0 to 31 and then goes to 0. Similarly, the triangle digital value is set from 0 to 15 and then back from 15 to 0. However, the number of bits required to be data is eight bits, but the sawtooth values is encoded to be 5 bits. Hence, three zero's are added after the encoding. Similarly, the triangle waveform value is encoded to be 4 bits, so four zero's are required to make total of 8 bits. Process of sending data bits out starts from setting `sync` to 0 at the falling edge of `sclk`. Then for each falling edge, the shift register does one left shift. The reason that the setting happens at falling edges because DAC reads a bit at rising edges and require some setting time (small but existing). Finally, when the 16th bit kept by a state machine is read, `sync` is brought high again.

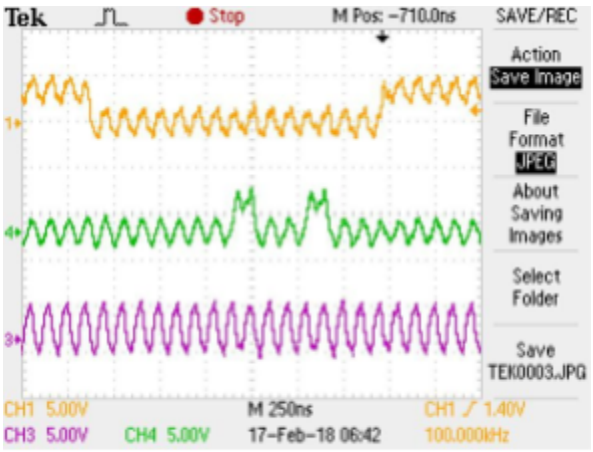
The results are shown in below figures including labels. In first three pictures, orange, green, and purple waveforms are `sync`, `mosi`, and `sclk` respectively, while the forth picture's purple waveform is the output of DAC. In the third picture, for example, it can be interpreted as `0b0000000010001000` of `0x0088`.



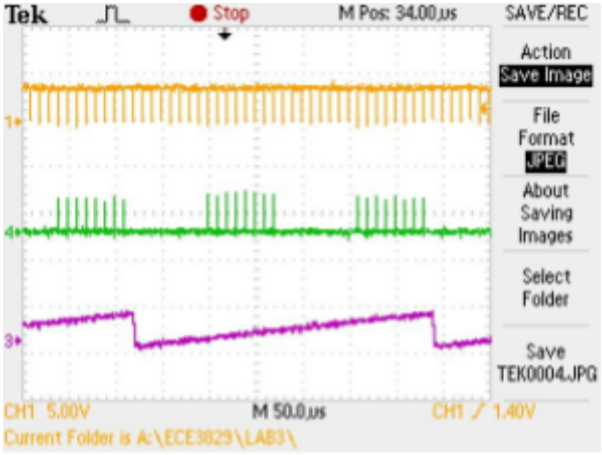
Signals from SPI when no switch is turned on



Signals from SPI when constant voltage switch is turned on

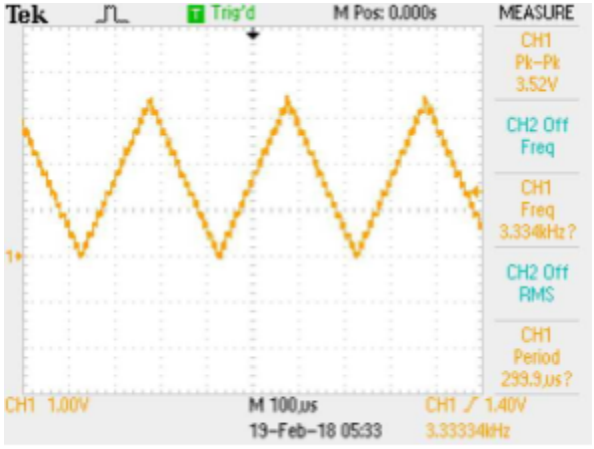


Signals from SPI when sawtooth waveform switch is turned on, which generates step voltages



Signals from SPI, `sync` and `mosi`, compared to output voltage from DAC

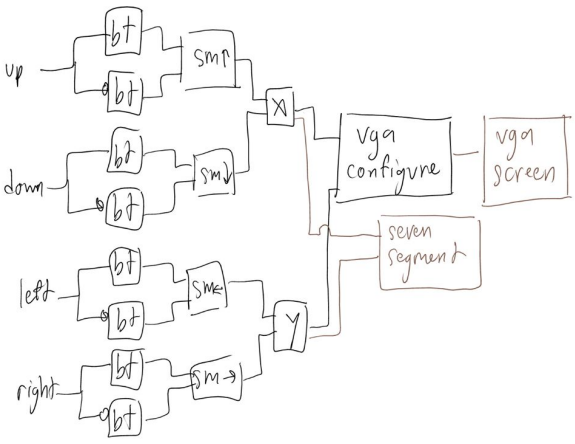
DAC output when sawtooth waveform is on

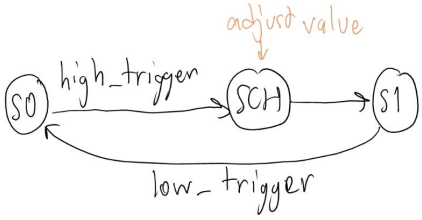


DAC output when triangle waveform is on

Part 2: Moving Block

In the block diagram, there are two more components other than vga configurer. First, ‘bt,’ which is supposed to mean ‘button trigger,’ is for handling button debouncing. The module is implemented to wait for long enough period of time for three times before it sends the signal back to indicate that the button is really pushed or unpushed. The next components are to keep the position of the block for each coordination. Initially, they are set to 9 because there is no exact middle for 20 separated blocks. The value changing is set through a state machine, shown on the right side.

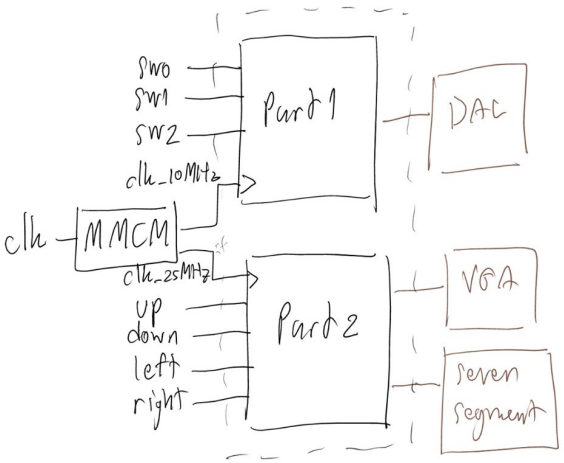




The state machine is initially at S0, indicating that the button is not pressed. Next, if the button is considerably really pressed (from the button trigger module), it will go to SCH, to apply the change to the variables; furthermore, the border condition is checked before each update to prevent overflow. Next, it proceeds immediately for the next clock period to S1, to prevent multiply applying changes. Finally, it will go back to S0, if the button is released in the similar fashion to when the button is pressed. In drawing the block, the positions on screen is divided by the width or height of the block to get the relative position. If the relative position equals to the current position kept by the variables, the bit is set to cyan. Otherwise, it will be white. In addition, the variables is printed to the seven segment. Note that the clock signal used is 25 MHz.

Part 3: Combination

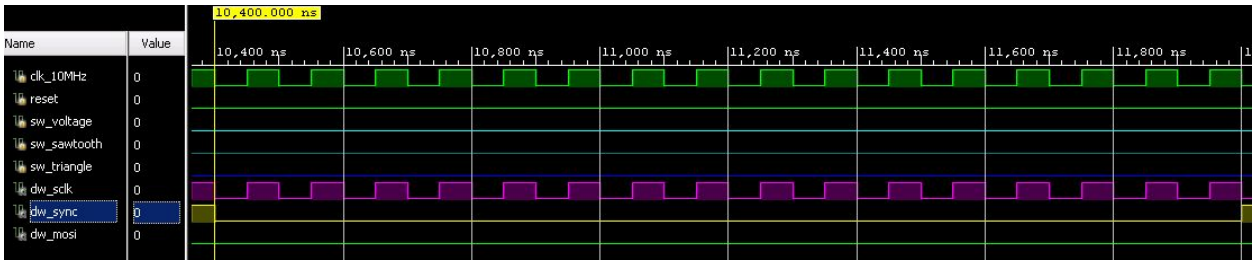
Because the modules are already created, they can be instantiated to be combined together. The seven segment for Part 1 does not clearly show the number, so it is used to print the positions on the VGA screen instead. Another point is the clocks used in both module are different. MMCM can generate two different frequency clock signals.



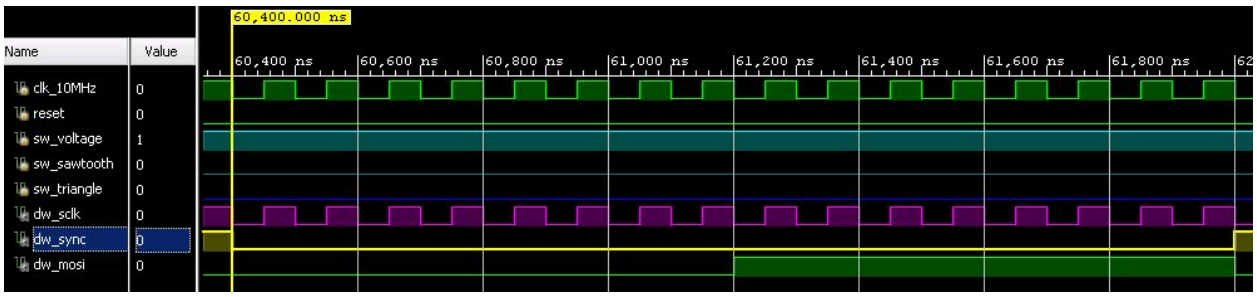
Part 4: Simulation and Testing

The simulation is used to test the correctness of SPI sent out by the DAC module. This simulation has the following processes. First, everything gets reset. Next, all inputs are disabled for a while. Then, the constant voltage is turned on for a while. After that, the sawtooth voltage is enabled for more than one cycle. Finally, the triangle voltage is enabled for more than one cycle. The names of variables may not be clearly seen. Purple, yellow, and green signals are `sclk`, `sync`, and `mosi` respectively.

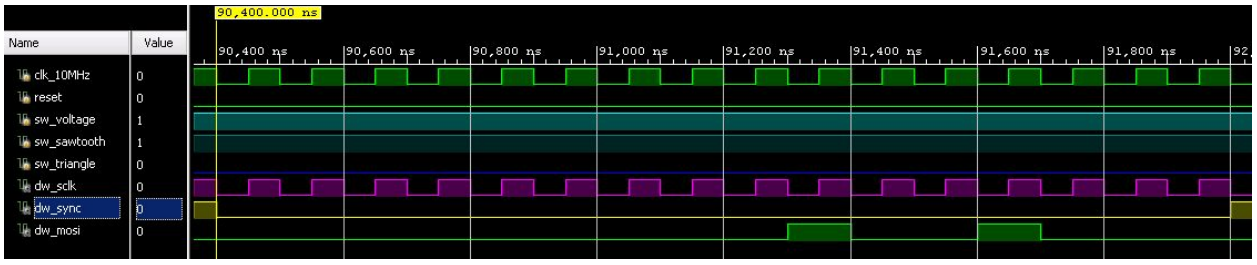
At first, zero voltage SPI is inspected; it is obvious that all values at the rising edges of `sclk` are zero as shown below. Notice that all switches are off.

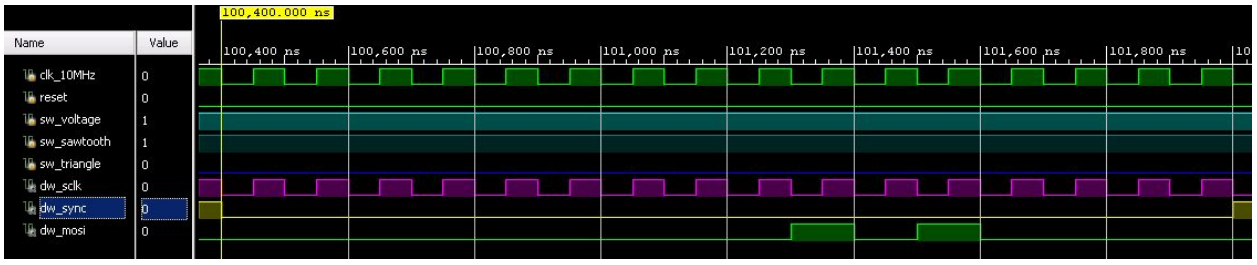


Next, the constant voltage switch is the only one that is on. The values excluding control bits should be all 1, which is shown in picture below.



Next, the sawtooth waveform is observed. In the first picture below, the value excluding the control bits traced starting at 90.4 μs at the rising edges of `sclk` is `0b01001000` or `0x48`, while in the second picture below (starting at 100.4 μs which becomes the next update), the SPI value is `0b01010000` or `0x50`. Those values are expected because each update should increase a previous value by `0b00001000` or `0x08`.





Similarly, the step of the triangle waveform is `0b00010000` or `0x10`. The first two pictures show the decrease step, while the last two pictures show the increase step. More precisely, The SPI data excluding control bits for the first two pictures are `0b11110000` or `0xF0` and `0b11100000` or `0xE0`. Also, the ones for the last two pictures are `0b00100000` or `0x20` and `0b00110000` or `0x30`. Note that the difference of starting times for both pairs is 10 μs. Those values are expected.

The Number of Flip-Flops

Utilization — **Post-Synthesis** | Post-Implementation — Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	343	20800	1.65
FF	240	41600	0.58
IO	35	106	33.02

Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	219	20800	1.05
FF	241	41600	0.58
IO	36	106	33.96
BUFG	3	32	9.38
MMCM	1	5	20.00

To make the count simple, flip flops are counted in coding sequence.

In `seven_seg.v`, the counter of clock divider uses 11 flip flops to count from 0 to 2000 to delay too high frequency clock for seven segment. The anode of seven segment uses 4 to easier handle the position on seven segment.

In Part 1, the counter of the clock divider uses 7 flip flops to count from 0 to 199. The value trackers of sawtooth and triangle waveforms use 5 and 4 flip flops to count from 0 to 31 and 15 respectively. The sign indicator of triangle waveform uses 1 flip flops; 1 is plus, while 0 is minus. The shift register and its machine state use 16 and 5 to keep data and to keep the number of data being rotated respectively. The data for seven segment use 16 flip flops to easier assign value to the seven segment module.

In `button_trigger.v`, the counter of clock divider uses 16 flip flops to count from 0 to 39999. The state of the state machine uses 2 flip flops to indicate how many cycle passed. The total of the module is 18 flip flops.

In Part 2, the positions (horizontal and vertical) of the block uses 5 flip flops for each (10 in total) to keep track number from 0 to 19. The state for each button uses 2 flip flops to indicate the status of button. The button trigger modules instantiated for eight times, which can be counted as 144 flip flops in total.

All parts can be combined to 214 that misses 6 - 7 more flip flops. It may be because there are some flip flops elements in the module that is created specifically for this lab.

Problem Encounter

The first problem is mismatching ports causing signal outputs to be out at wrong ports. This could be simply solved by swapping them back.

Next, the seven segment in DAC module is not readable when running with sawtooth waveform or triangle waveform because the value changes faster than human eye can recognize; it cannot be solved and is left.

For VGA module, we encountered with debouncing issue, so a new module, `button_trigger`, is implemented to assist the module to decide whether the button is really pressed or unpressed. All buttons, both pressing and releasing, are parts of inputs of instantiated modules. Nevertheless, it seemed the module still produced similar error at first, but after increasing the period of waiting time, we can get a correct result.

One warning that occured in the synthesis process is inferred latch. It is thrown because the use case of a state machine did not cover all the possible case generated regularly even though it could not go to that state. We solved it by adding default option for case which goes to an initial state.

Last, a simulation did not work the way we expected and produced ‘X’ as outputs. we found out that the module initially did not assign any values to some ports; the simulation could not assume that those values started at some values even though it is simply a counter. we solved this error by adding initial value to variable declaration.

Warning

The picture above shows all warnings from synthesis and implementation process. The first one refers to value assigning which happens only the change of panel selecting; it is not necessary to change assignation for each change of values. The second (and three) refer to the missing connection which is meant to connect to 0. The fourth and fifth refer to unused values, but the values are used to indicate the status of those values that are related in the modules.

Conclusion

We can accomplish in designing and implementing circuits to DAC and VGA display. Precisely, we learn how to send SPI data to expectedly generate correct waveform via DAC. Moreover, we can implement separate module to correctly handle button debouncing and produce acceptably block-moving VGA display controlled by buttons. As usual, many troubles occurred during the process. Unexpected values while simulting is one of the big problem. Initialization can solve this problem. Also, it reflects that a better reset functionality should be in a module regularly. In conclusion, we can successfully design electronic circuits to generate waveforms and create appropriate system to move a block on VGA through buttons without unpredicted situations. In the future, we can add separate input to to be able to control either frequency or maximum peak of waveforms.

Appendix