**CMPE-250 Laboratory Exercise Twelve:**

**D/A Conversion, A/D Conversion, PWM, and Servos**

**with Mixed Assembly Language and C**

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students; however, other than code provided by the instructor for this exercise, all code was developed by me.

John Judge

Performed 4/28/16

Submitted 5/5/16

Lab Section 02

Instructor: Dr.Shaaban

TAs: Peter Muller

Stephen Moore

Connor Goss

Lecture Section 01

Professor: Allesandro Sarra

**Abstract**

In this exercise investigated the use of transferring data between different domains using the Freescale Freedom Board KL46Z. The objective of the exercise was to implement a program capable of analog-to-digital (A/D) and digital-to-analog (D/A) conversions as well as pulse-width modulation. The exercise tested the knowledge and use analog and digital measurements. The program written in this exercise was able to convert a character received through a peripheral device to the desired position of a servomotor. The process was done using a look-up table to convert the character to a digital value. The digital value was then converted to an analog value. The analog value, which was in volts, was then converted back to a digital value. The final digital value was then in turn converted to both the original ASCII character and a digital value representing a servo position. The character was then echoed to the terminal and the digital value was converted to a value the servomotor could read and adjust itself to. This process was achieved using the DAC0, ADC0, and TPM0 connections on the KL46Z board. In this exercise multiple C functions needed to be developed for these data conversions as well as a main function to drive them. The program was created to specifications desired in the exercise and was tested using an Oscilloscope and Futaba S3003 servo. The exercise was successful and verified through testing. Through this exercise several lessons were learned and were reflected on.

**Procedure**

To begin the exercise all Assembly Language subroutines that would be used during the exercise were added to the Assembly source file. No new Assembly Language subroutines were implemented during the exercise. All were developed during previous exercises and included: *AddIntMultiU, UART0\_ISR, PutNumHex, InitQueue, Enqueue, Dequeue, PutChar, GetChar, GetStringSB, PutStringSB, LengthStringSB, Init\_UART0\_IRQ,* and *PutNumU*. These subroutines, as well as there respective equate statements and variables, were added to the project. In order to achieve the connection with the C source file that would be developed later in the exercise, the necessary export statements were needed for the subroutines that would be called in the C source file. A list of these subroutines was found in the C header file and the export directives were addressed immediately preceding the subroutines in the “MyCode AREA.” These subroutines provided the basis for the polled serial I/O with the peripheral device that would be used for user interface as well as some useful functions that eliminated the need to rewrite their equivalent C functions.

Once the basics for the exercise had been added, the look-up tables (LUT’s) referenced in the C header file needed to be added to the constants area of the Assembly source file. In hardware LUT’s provide a fast and space efficient way to preform calculations. Once described in software they can be configured on silicon by synthesis tools. In this exercise the Keil IDE was able to preform the synthesis to implement *DAC0\_table\_0* and *PWM\_duty\_table\_0* on the KL46Z board. These LUT’s were used to convert the ASCII character from the terminal to-and-from a digital value and a digital value to a one that could represent a servo position, respectively. By adding them to the constants area defined by AREA MyConst they could be added to the KL46Z’s ROM and would be available to preform their respective operations without writing a function to preform the conversions based on an algorithm.

Next the C source file needed to be created to handle the operations described by the objectives and specifications of the exercise. Within the C source file functions were created to handle the operations of the digital-to-analog and analog-to-digital conversions as well as the main function used to drive the program. In this file the functions to both initialize the DAC0, ADC0, and TMP0 connections and to handle the data conversions in the process described by the exercise specifications.

The first function written in the C source file was *Init\_DAC0*, which initialized the DAC0 connection on the KL46Z board. The port was initialized by first enabling the TPM0 module clock and port E module clock. The DAC0\_OUT was then connected to port E pin 30 on the microcontroller. The DAC0 and DMA were then disabled. Finally the DAC0 was then enabled as a reference voltage and interrupts were disabled. When run, *Init\_DAC0,* enabled the DAC0 port on the microcontroller. The ADC0 port could then reference the voltage of this port.

Next the function *Init\_ADC0* was created to initialize the ADC0 port. The ADC0 port was initialized and calibrated for polled conversion of the single ended channel AD23. This port was able to receive a reference voltage from the DAC0 port in the range of 0 V to 3.3V. This voltage was then converted to an integer in range from 0 to 1023 by the function *getADCConversion.*

In order to assign the voltage at the DAC0 port the function *writeToDAC0* was written. This function converted the character received by the peripheral machine providing the user interface to the system to a voltage at the DAC0 port. This was done by converting the ASCII character to its decimal equivalent. This value was then converted to another integer value by the LUT *DAC0\_table* in the range of 0 to 4095. This value was then converted to a voltage by preforming a bitwise AND of the integer and hexadecimal FF to mask all bits other than the most significant 8 bits. The result was then shifted to the 8 least significant bits and then signified a voltage in range 0 V to 3.3 V. This function essentially drove the conversion from ACII-to-digital and digital-to-analog.

The function *getADCConversion* was written to convert the analog value calculated by *writeTODAC0* to a digital value for the final LUT. This was achieved by assigning the voltage at the ADC0 port to source channel one. This value was then returned when a change in voltage was detected.

The functionality to convert the digital value returned by *getADCConversion* into a position of the servomotor was written in *calcServoPosition*. This function simply multiplied the return of *getADCConversion* by five and divided the resulting value by 1024. This value then had to be increased by one to correctly index the *PWM\_duty\_table\_0* LUT.

A final function needed to be written to access the *PWM\_duty\_table\_0* LUT and return and return the desired servomotor position. Thus the function *moveServo* was written to preform the conversion by driving the LUT. This was achieved by accessing the *PWM\_duty\_table\_0* LUT at the position passed to it by *getADCConversion* and returning the value of the LUT. Following the creation of *moveServo* the main function was implemented to order the flow of data between the functions.

The main function essentially declared the flow of data between all of the C functions and Assembly Language subroutines thus creating program the user could interface with through the peripheral. In order to initialize the components of the program all interrupts needed to be disabled and the *Init\_DAC0, Init\_ADC0,* and *Init\_TPM0* functions need to be called to initialize the connections on the microcontroller. Next the subroutine *Init\_UART0\_IRQ* was called to initialize the transmit and receive queues to interact with the user just as previous exercises did. Once everything was initialized the function operated in an infinite for loop in which the user prompted to provide a desired servo position. If a character received was in the acceptable range the program would echo the character and convert it to a digital value by *writeToDAC0* and print this value to the terminal. Then the value was converted to an analog value by *getADCConversion* and also printed to the terminal. The servomotor position was then calculated and moved by calling *calcServoPosition* and *moveServo* respectively. The servo position was then printed to the terminal using and the infinite for loop would repeat. Following the completion of the main function the program was then tested using an Oscilloscope and Futaba S3003 servo.

**Results**

After the program was written it was tested. In order to ensure the program was written correctly an oscilloscope probe was attached to the TPM0\_CH4 connection on the microcontroller and the UART0 port was connected to the peripheral machine. Once the program was run the prompt appeared on the terminal as in Figure (1.0). The program was then tested using each of the five possible servo positions. Starting from the initial width of the digital signal, Figure (1.1), the width decreased for each increasing number entered into the terminal from ‘1’ to ‘5’ (Figure (1.2) – Figure (1.6), respectively). These were the correct results for each of the cases. The results displayed on the terminal were verified to be correct. It was also during this time that the program was tested by entering invalid servo positions and the program responded correctly by waiting until a valid position was entered. Following this testing, the microcontroller was ready to be tested with the Futaba S3003 servo.

In order to test if the program would successfully drive the servo to the desired positions connections need to be made from the microcontroller to the servomotor. The TPM0\_CH4 port of the microcontroller was connected to the control port of the servo. Next the ground port of the servomotor was connected to the ground of the microcontroller. In order to power the servo a 5 V DC signal from the DC power supply was needed. A connection was made between the power supply to the servo and from the ground of the power supply to the ground of the microcontroller. This ensured that all the equipment had a common ground reference node. The same tests occurred that were used in Figure (1.0) but at the time of testing the tests produced incorrect results.

Although the terminal read the correct results, the servo did not perfectly align to the desired positions. The servo positioned itself slight too far clockwise of the desired positions. In order to correct this the equate for the *WM\_DUTY\_10* in the Assemble Language source file was changed to be 5500 instead of 6000. This would position the servo slightly more counterclockwise than it previously was. The tests were then rerun and the servo aligned with the desired positions indicating that the objective of the exercise was met.

**Conclusion**

In this exercise investigated the use of transferring data between different domains using the Freescale Freedom Board KL46Z. The objective of the exercise was to implement a program capable of analog-to-digital (A/D) and digital-to-analog (D/A) conversions as well as pulse-width modulation. The program written in this exercise was able to convert a character received through a peripheral device to the desired position of a servomotor. The process was done using a look-up table to convert the character to a digital value. The digital value was then converted to an analog value. The analog value, which was in volts, was then converted back to a digital value. The final digital value was then in turn converted to both the original ASCII character and a digital value representing a servo position. The character was then echoed to the terminal and the digital value was converted to a value the servomotor could read and adjust itself to. This functionality was verified to be correct by examining the pulse widths generated with an oscilloscope as well as verifying the servo reached the correct positions and the terminal displayed correct results. The program preformed as expected and all the results were verified to be correct. As a result, the objectives of the exercise were met. The exercise was determined to be successful because the program operated as expected and the use of mixed assembly language with C as well as the desired data conversions was correct. The implementation and use of the both of these proved that the theory had been learned. During this exercise device programming in C and assembly language was achieved. It was during the implementation that use of C was determined to be far more convenient compared to assembly language. Although C does not allow for as much control over the movement of data through the device it is far more convenient and less time consuming to code in C. As a result all new functions written in this exercise were written in C. The stark contrast between the two languages will certainly be something noted during future exercise and projects.

**Appendix**

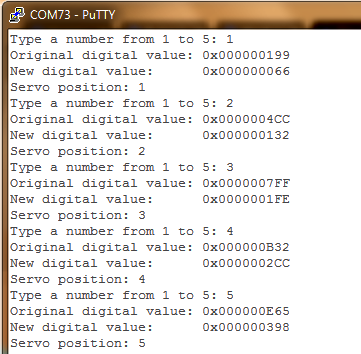
****

Figure (1.0): The Terminal Results

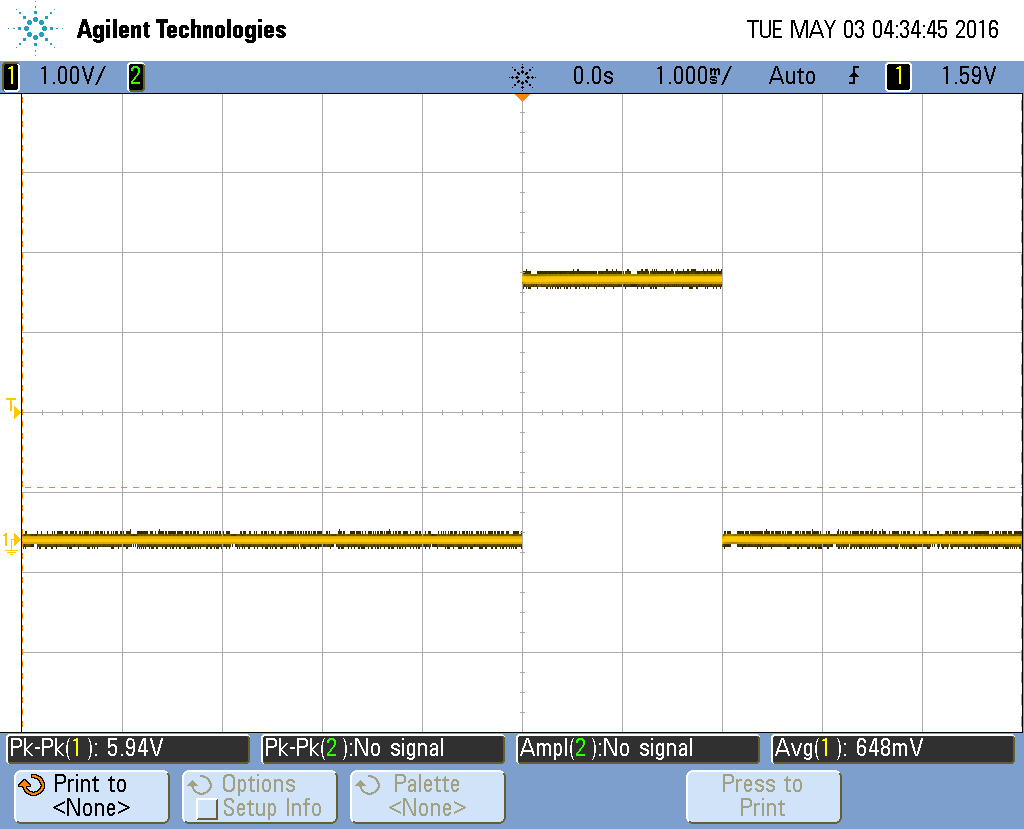


Figure (1.1): The Initial Condition of the Pulse Before a Desired Servo Position Had Been Received

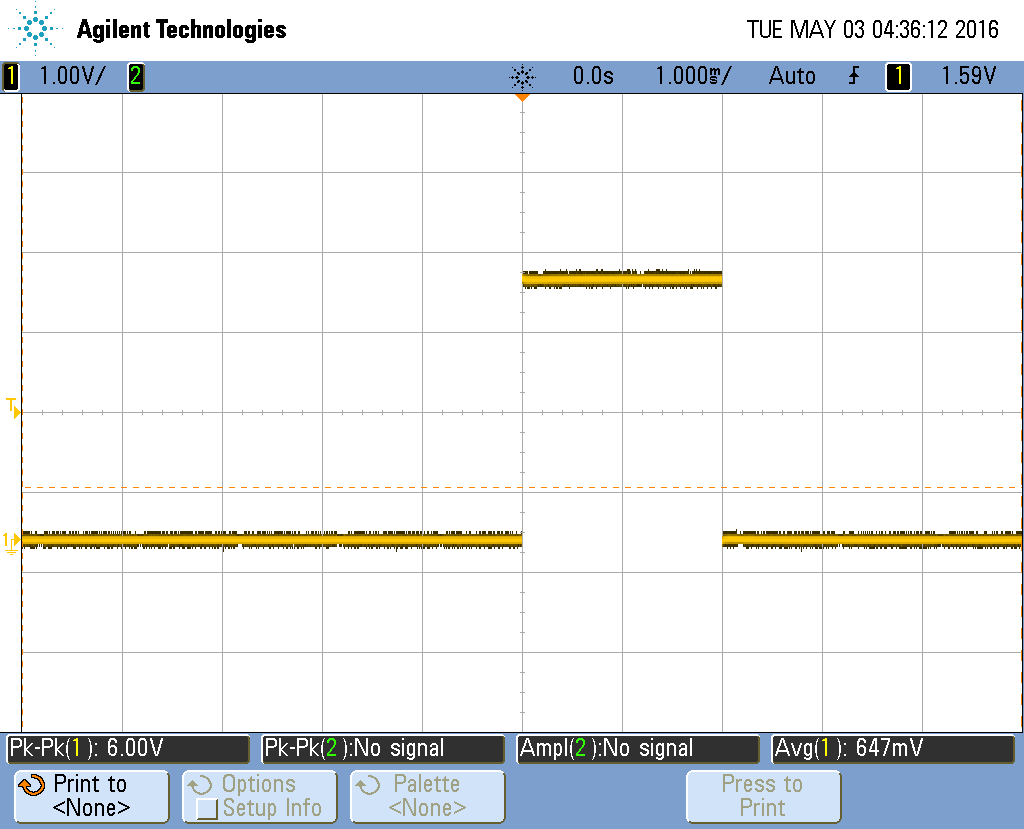


Figure (1.2): The Change In Pulse Width When Servo Position ‘1’ Was Requested

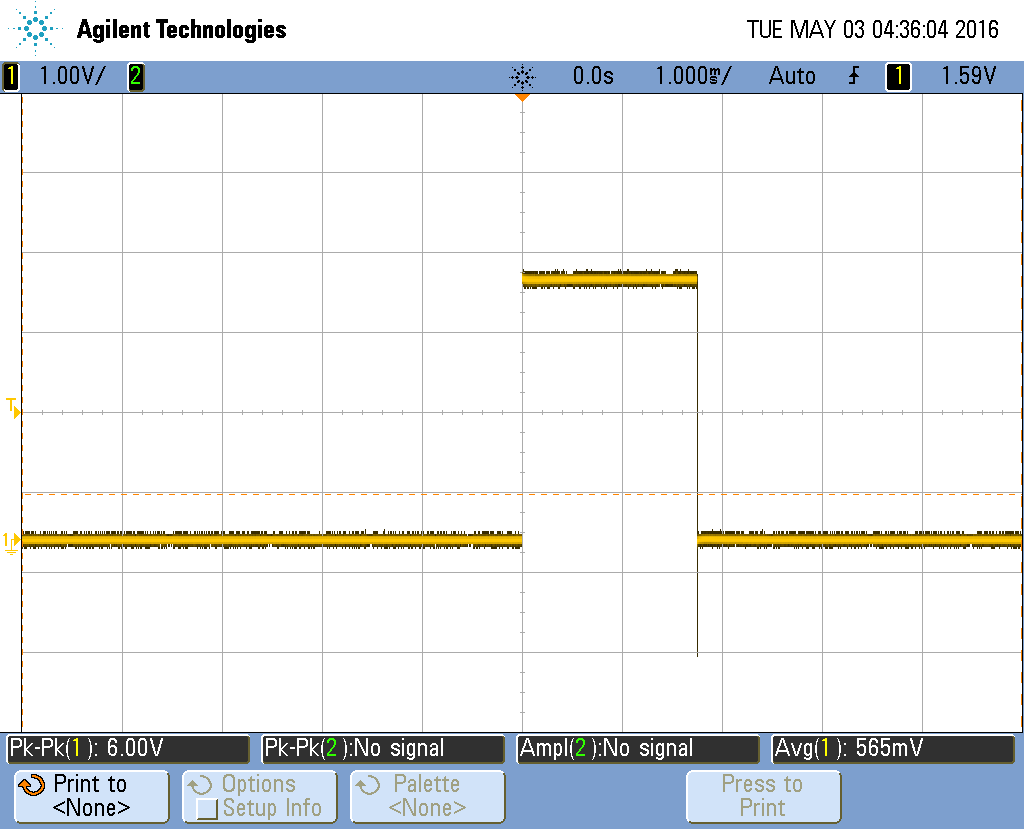


Figure (1.3): The Change In Pulse Width When Servo Position ‘2’ Was Requested

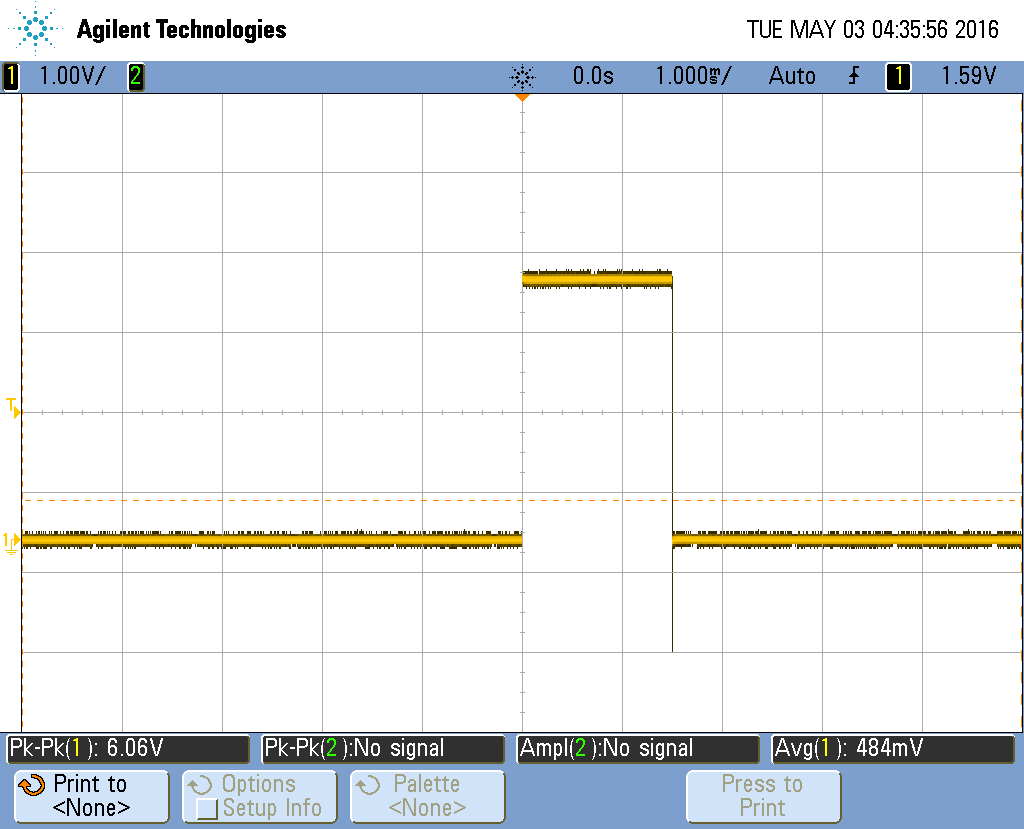


Figure (1.4): The Change In Pulse Width When Servo Position ‘3’ Was Requested

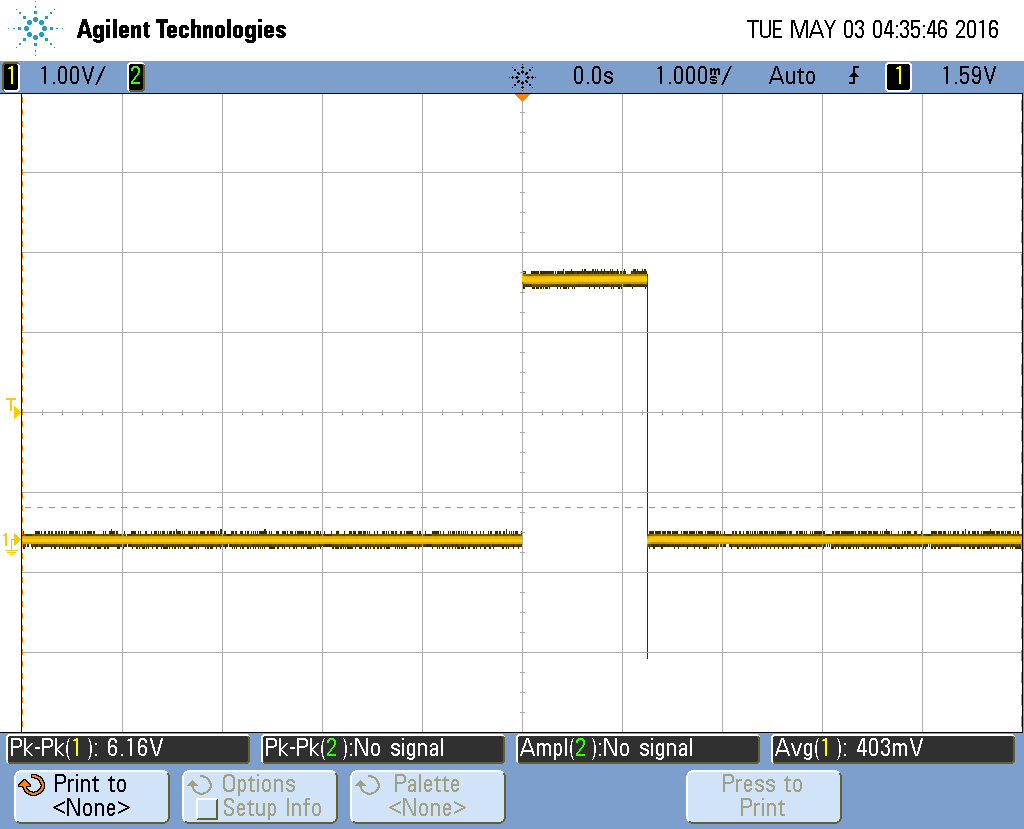


Figure (1.5): The Change In Pulse Width When Servo Position ‘4’ Was Requested

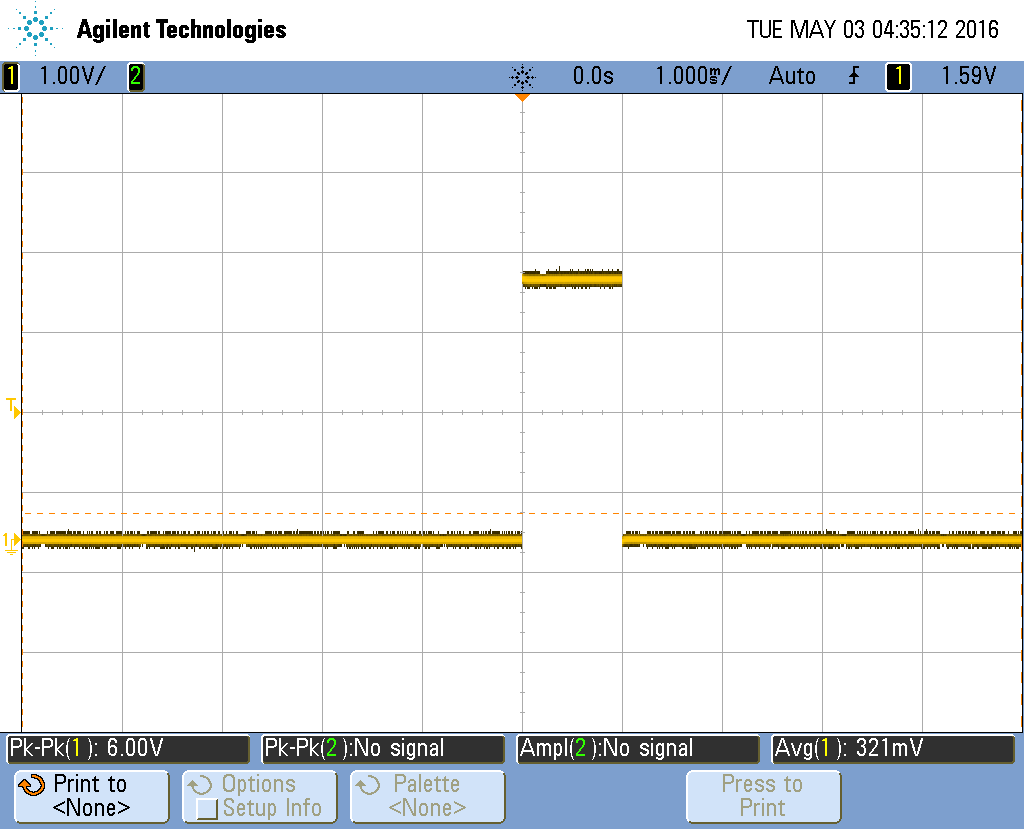


Figure (1.6): The Change In Pulse Width When Servo Position ‘5’ Was Requested