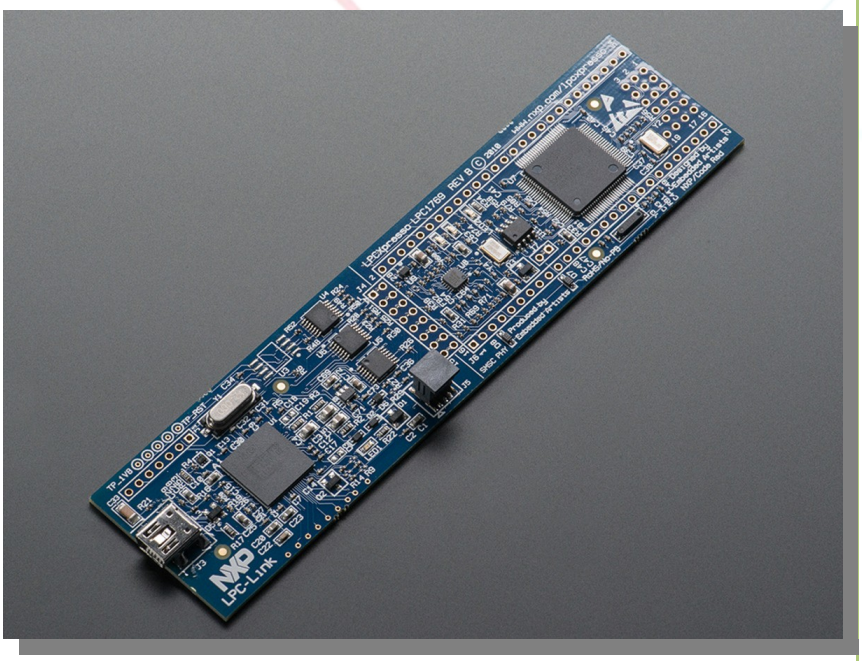
TENET TECHNETRONICS

2016

Interfacing Servo motor with NXP LPC1769 using LPCXpresso



Author: Gurudatta Palankar

Reviewers:

Version: 1.0

9/3, 2nd floor, SreeLaksmi Complex, opp, to Vivekananda Park, Girinagar, Bangalore - 560085,
Email: info@tenettech.com, Phone: 080 - 26722726

Introduction:

LPCXpresso™ is a new, low-cost development platform available from NXP supporting NXP's ARM-based microcontrollers. The platform is comprised of a simplified Eclipse-based IDE and low-cost target boards which include an attached JTAG debugger. LPCXpresso™ is an end-to-end solution enabling engineers to develop their applications from initial evaluation to final production.

Step 1: Open LPCXpresso IDE

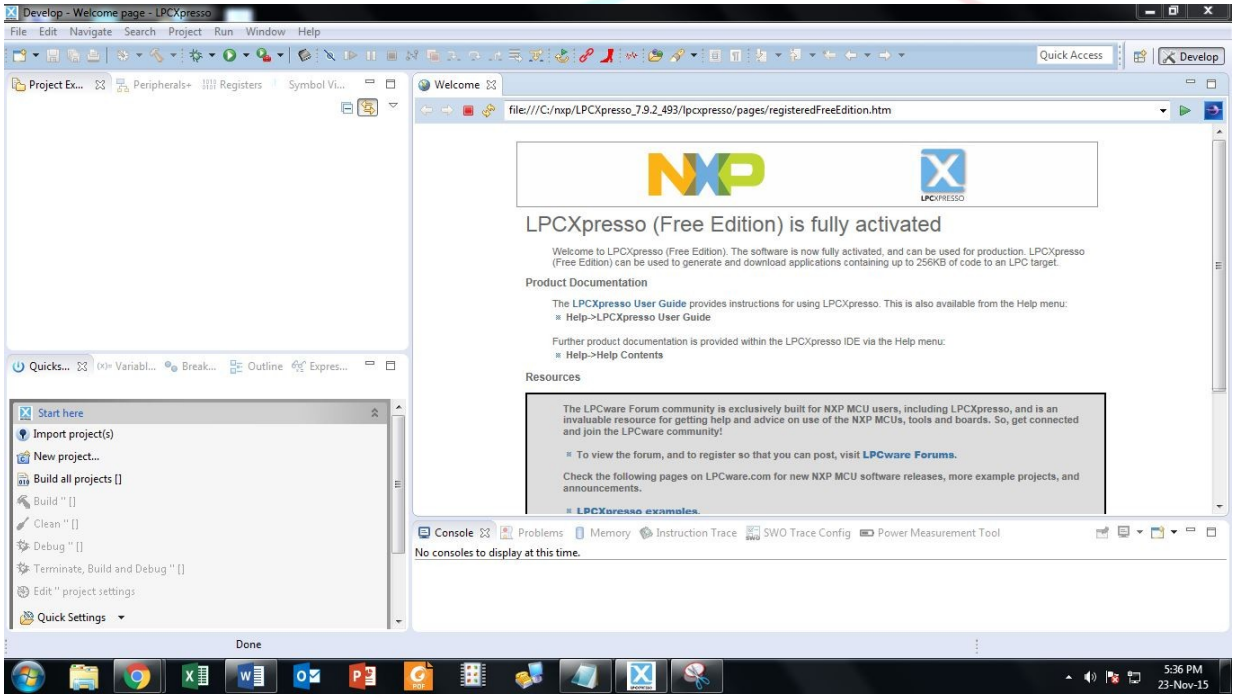


Figure 1

Step 2: Before writing a code, we have to Import some Library Files to the Workspace. Click on Import projects on Quickstart Panel on the bottom left of the window.

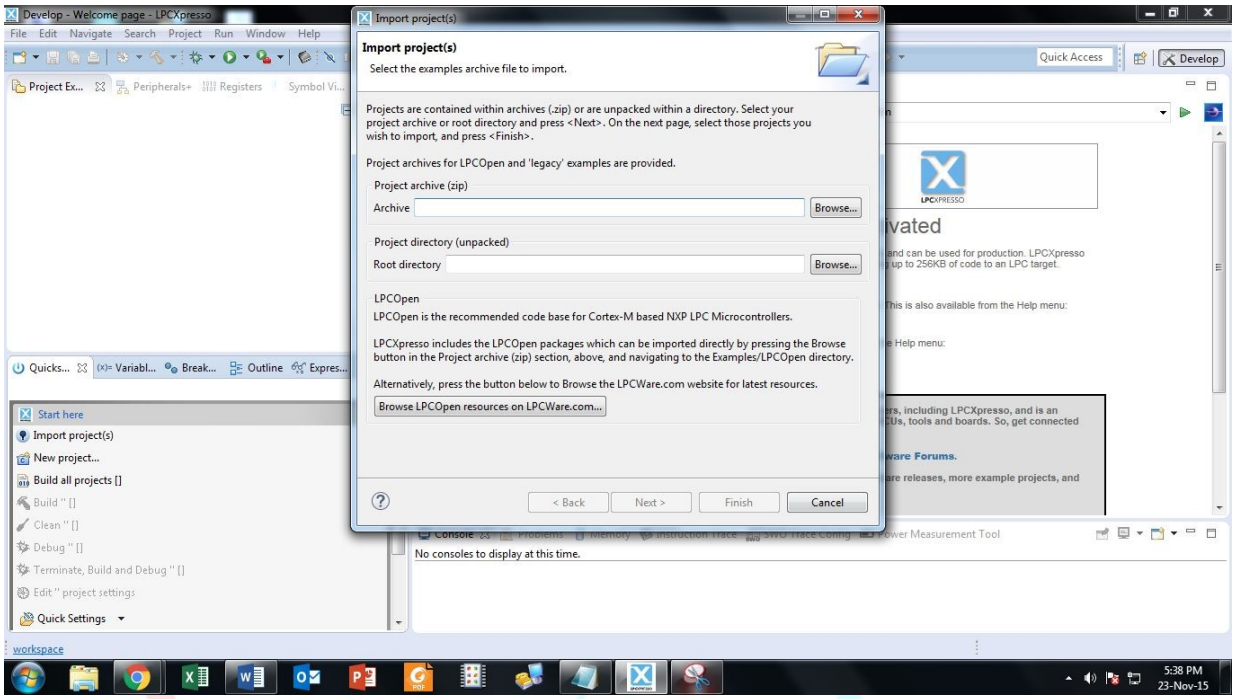


Figure 2

Step 3: Browse file, open the LPC1000 folder.

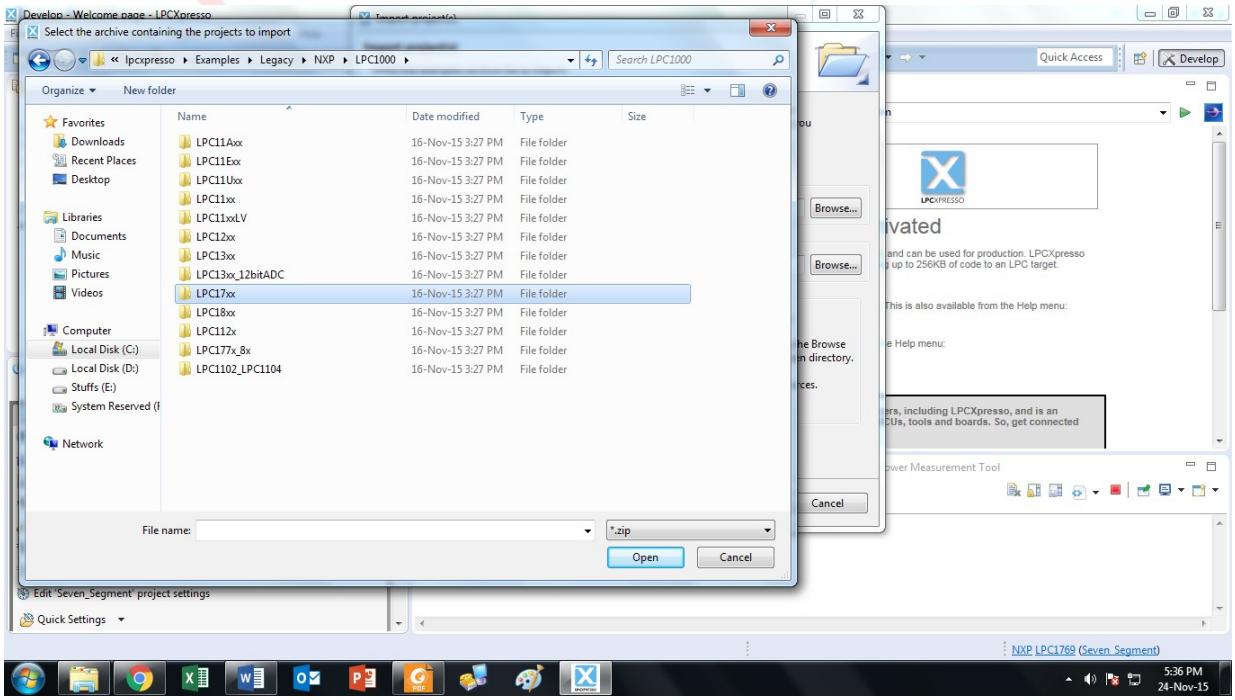


Figure 3

Step 4: Select the appropriate archive file. Let us select LPCXpresso176x_cmsis2. We can select CMSIS CORE library that include LPC17xx.h header file.

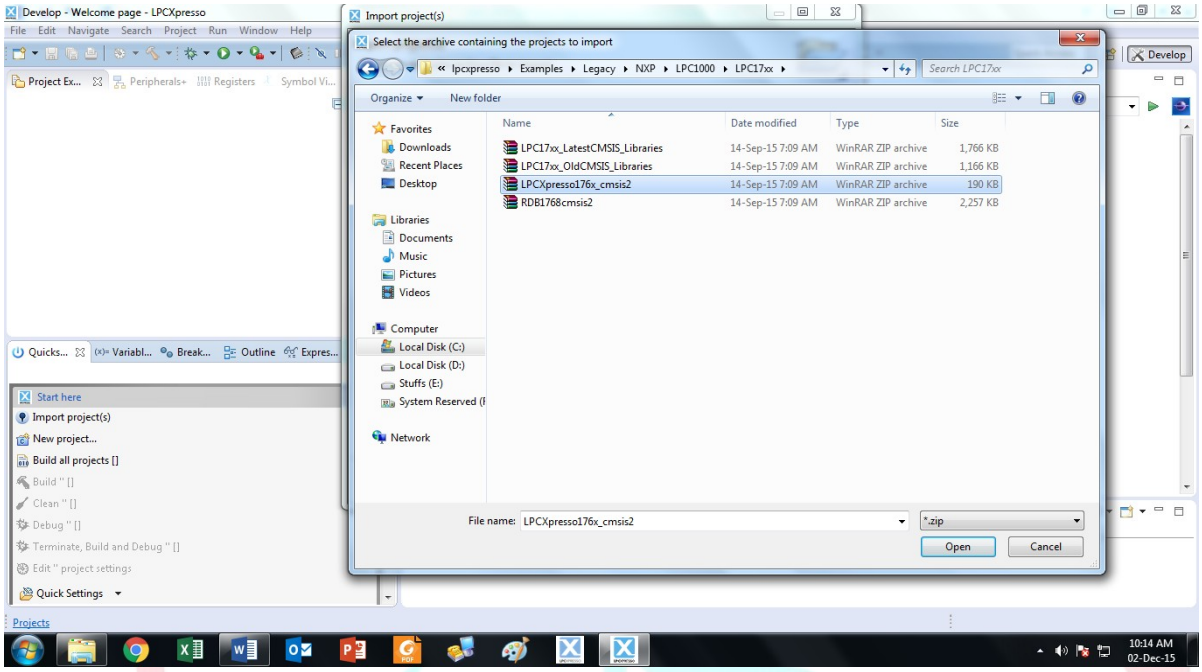


Figure 4

Step 5: After selecting you will be able to see the following libraries files. Let us select following libraries. Here we are using LPC176x_cmsis2_systick to have precise time delay.

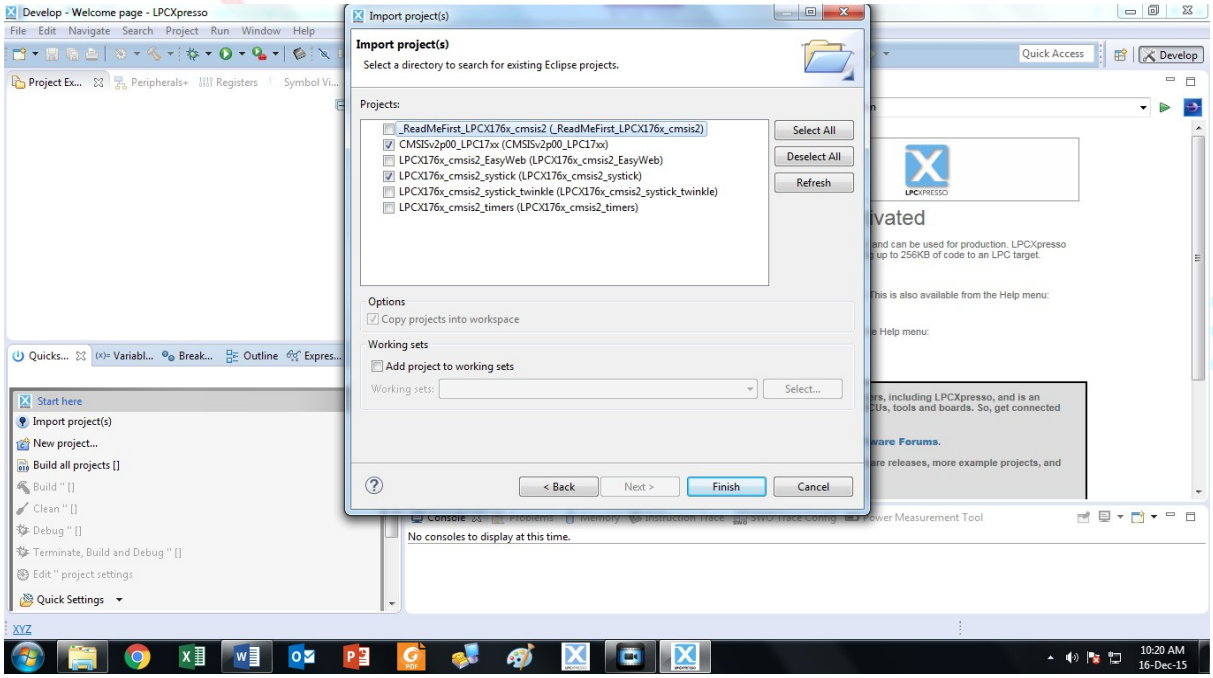


Figure 5

Step 6: Now we will be able to see those libraries in the workspace.

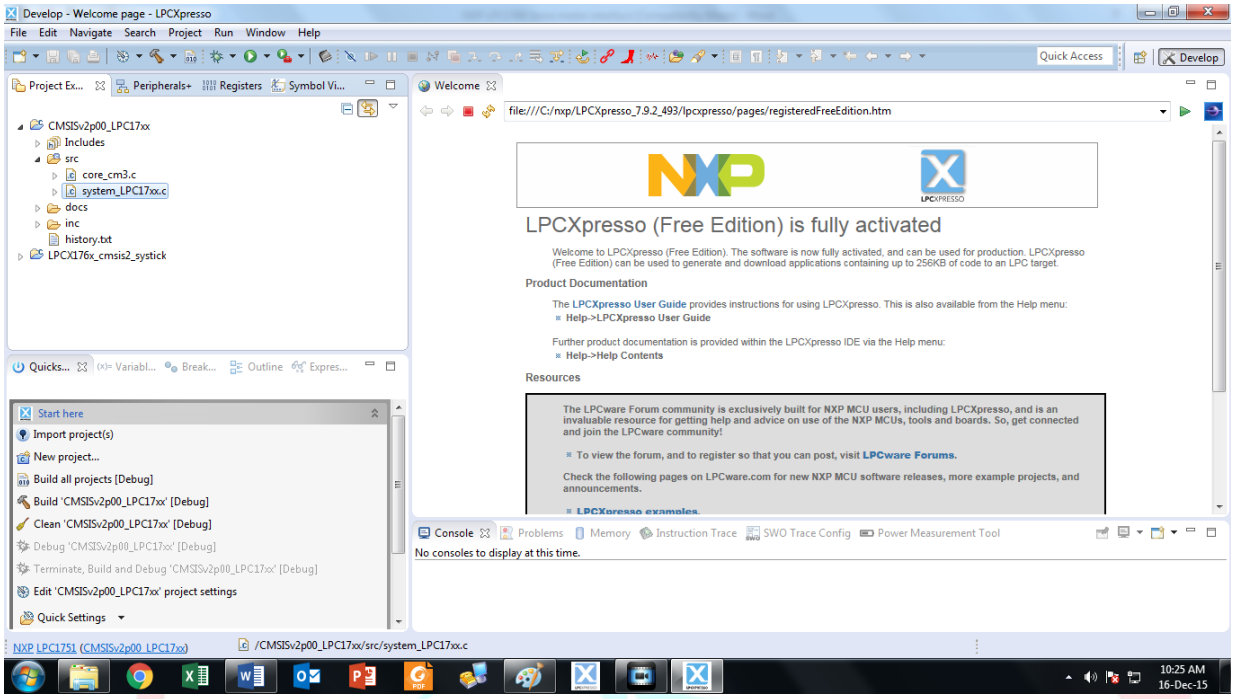


Figure 6

Step 7: Now we can start creating our new project. Goto File >> New >> Project. Select LPCXpresso C project.

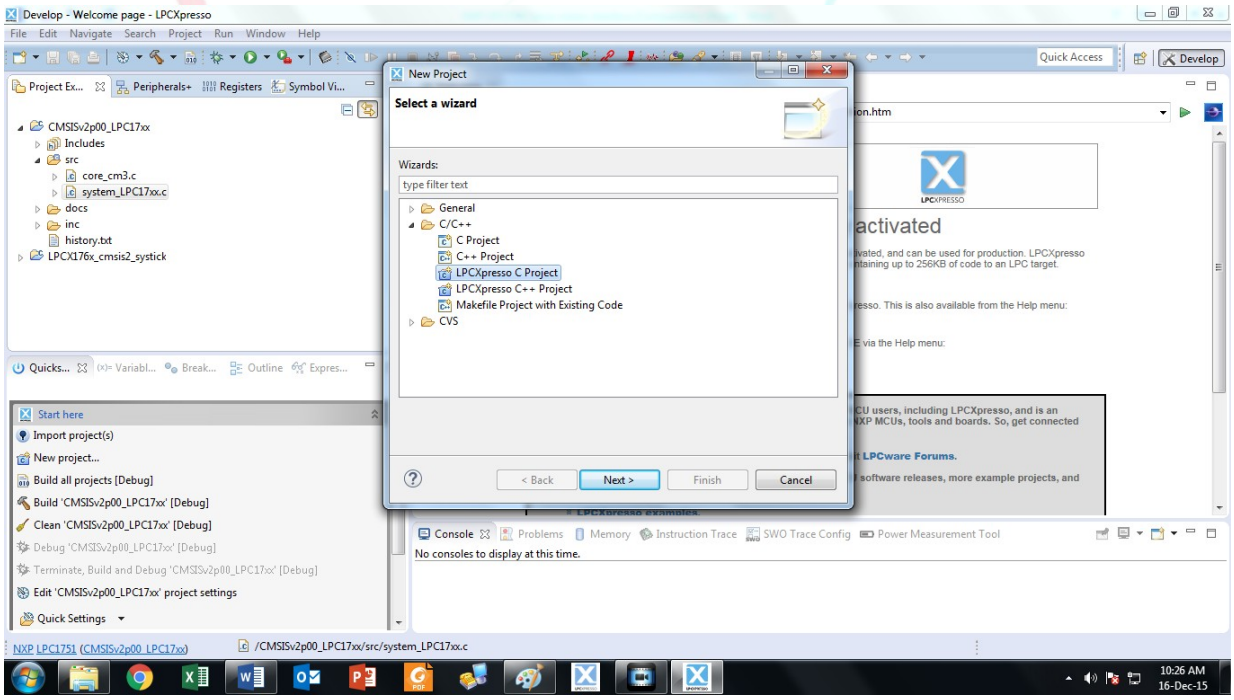


Figure 7

Step 8: Select LPC1769, C Project and give name to your project. Select target MCU as LPC1769.

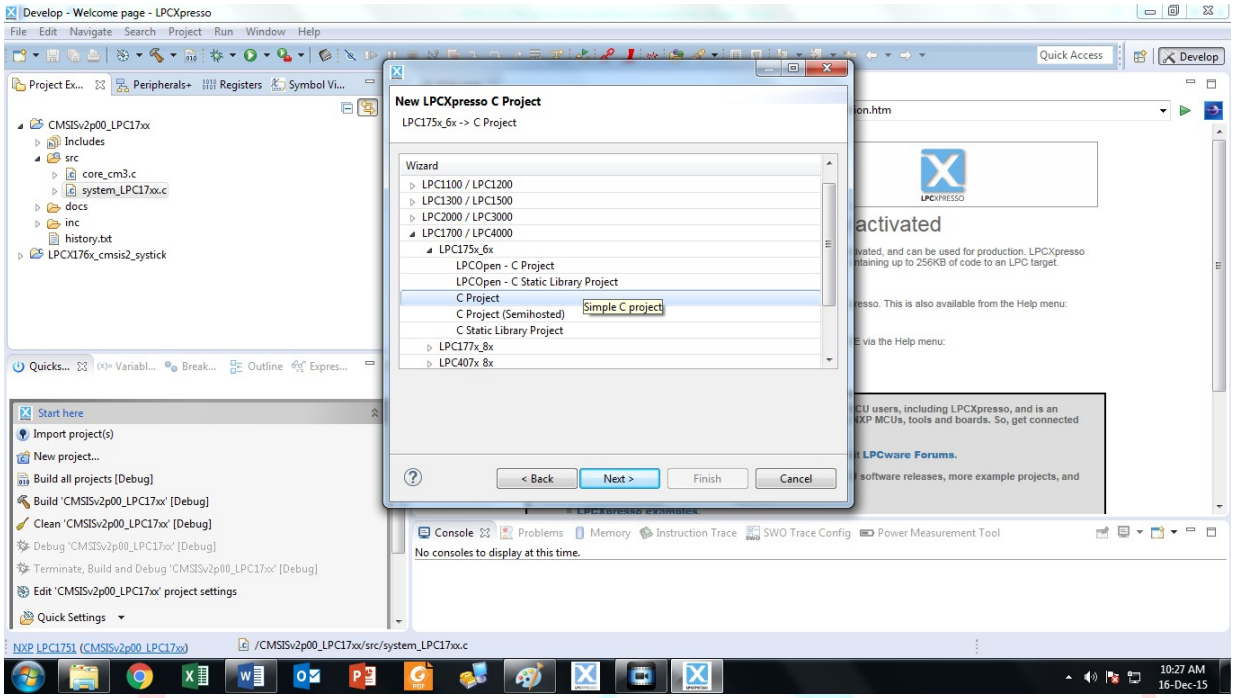


Figure 5

Step 9: Now select CMSIS Core library. Click on Next and keep all the other configurations as default and Finish.

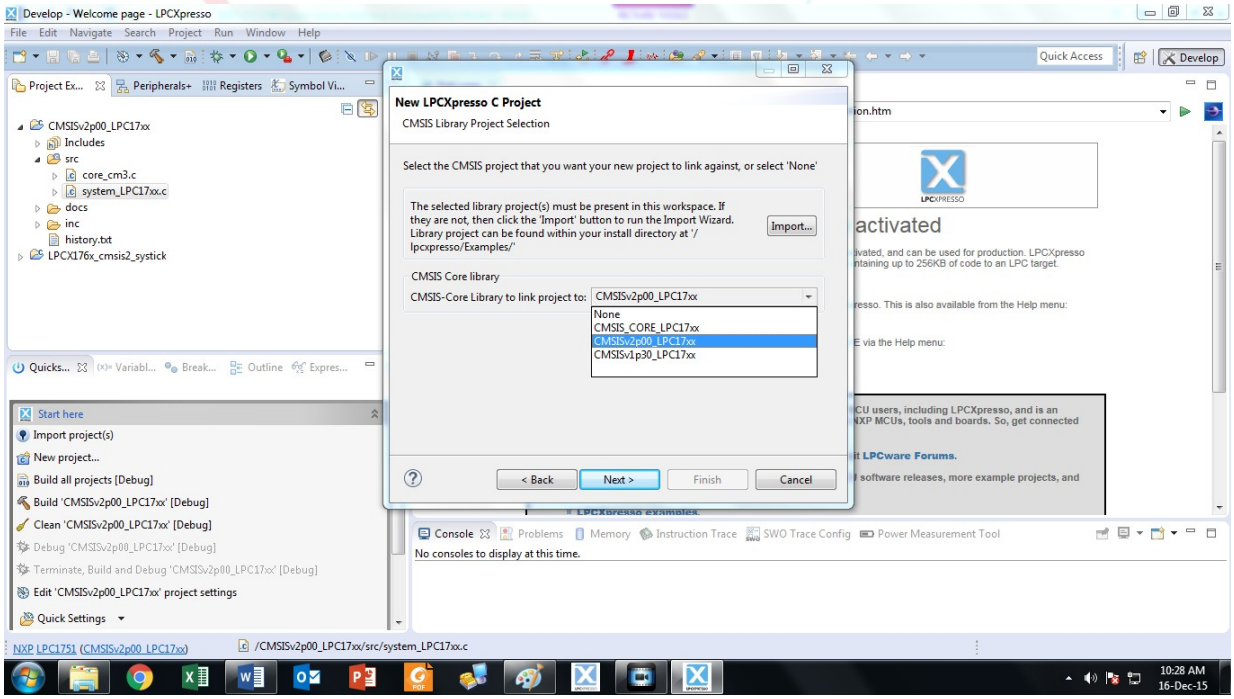


Figure 9

Step 10: Now we can see our project onto the workspace. Now by double clicking on Servo_motor.c file, we can start writing code.

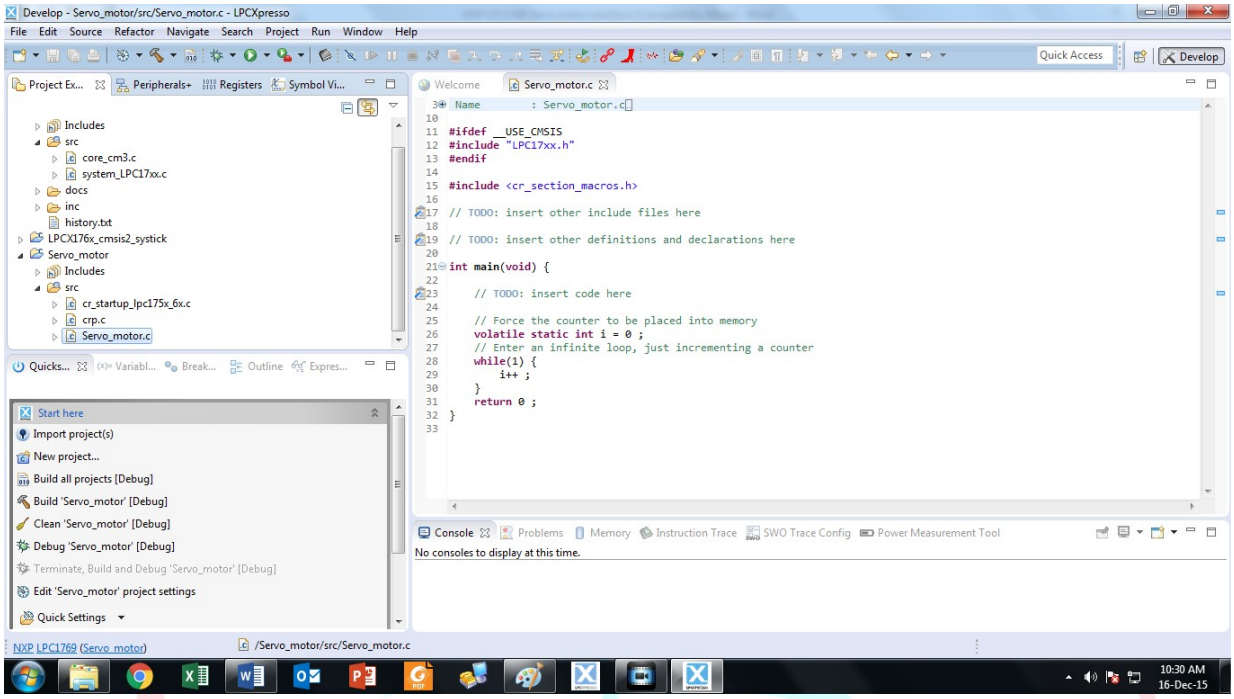


Figure 10

Step 11: Before writing code create a header file for delay in the Project file. To do this, double click on systick_main.c and copy the code. Do not copy the main() program.

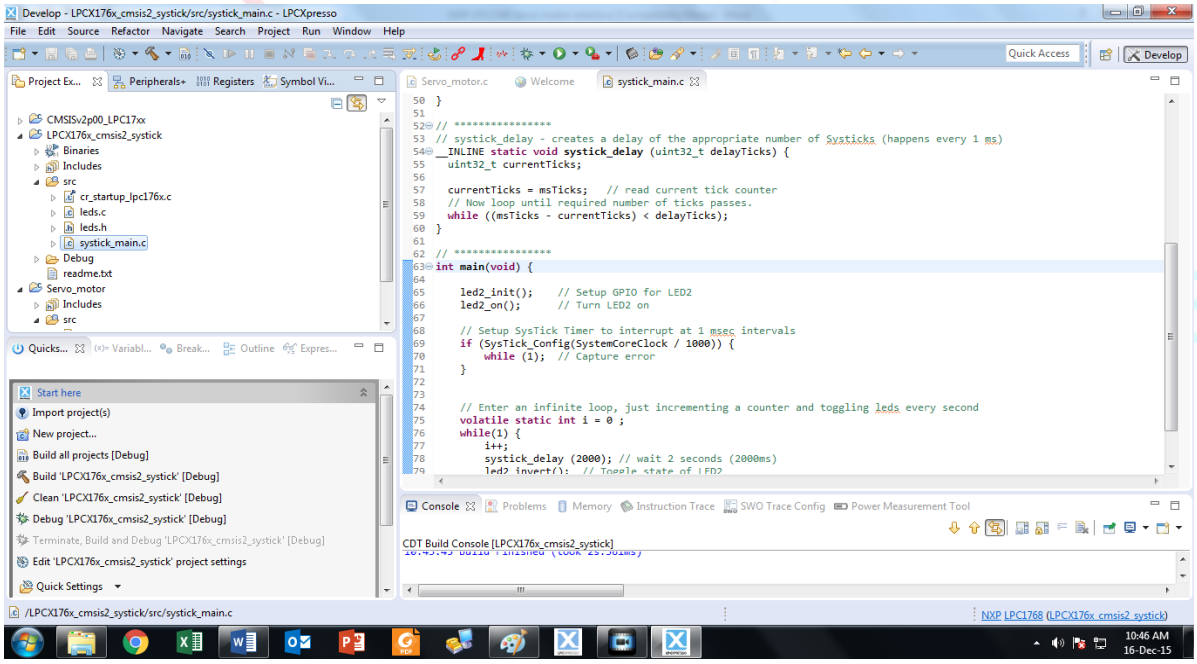


Figure 11

Delay Subroutine:

```
#include "LPC17xx.h"
#include <cr_section_macros.h>
volatile uint32_t msTicks;  //counter for 1ms SysTicks

void SysTick_Handler(void) //SysTick_Handler- just increment SysTick counter
{
  msTicks++;
}

/* systick_delay - creates a delay of the appropriate number of Systicks
(happens every 1 ms) */
__INLINE static void systick_delay (uint32_t delayTicks)
{
  uint32_t currentTicks;
  currentTicks = msTicks;                       //read current tick count
while ((msTicks - currentTicks) < delayTicks);  /* Now loop until required
number of ticks passes */
}
```

Step 12: To create header file in project, right click on Servo_motor project file then, New >> Header file.

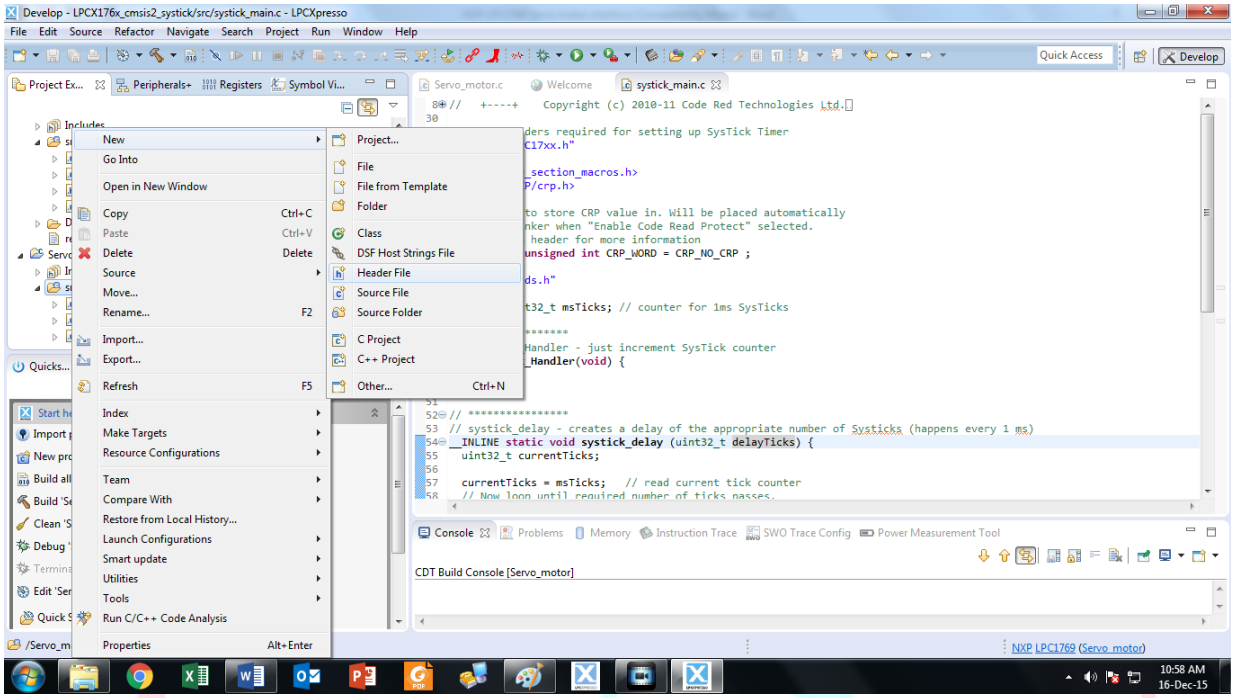


Figure 12

Step 13: Give header name with .h file extension.

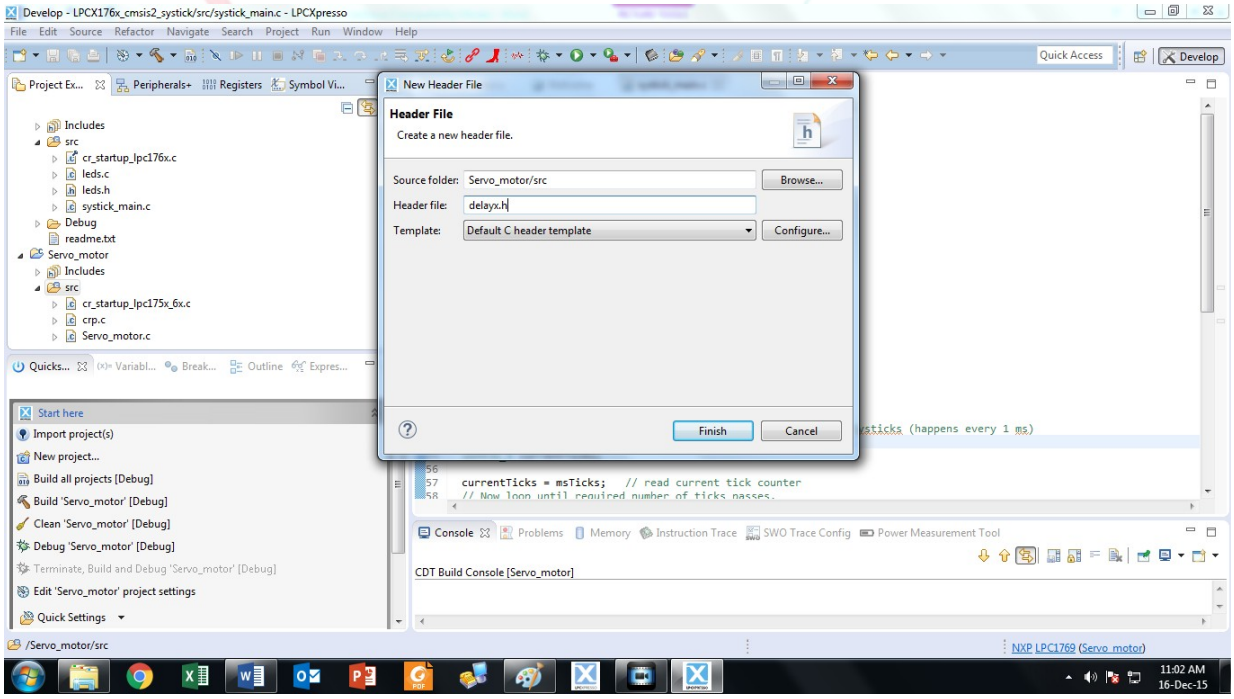


Figure 13

Step 14: Paste the delay subroutine in the header file.

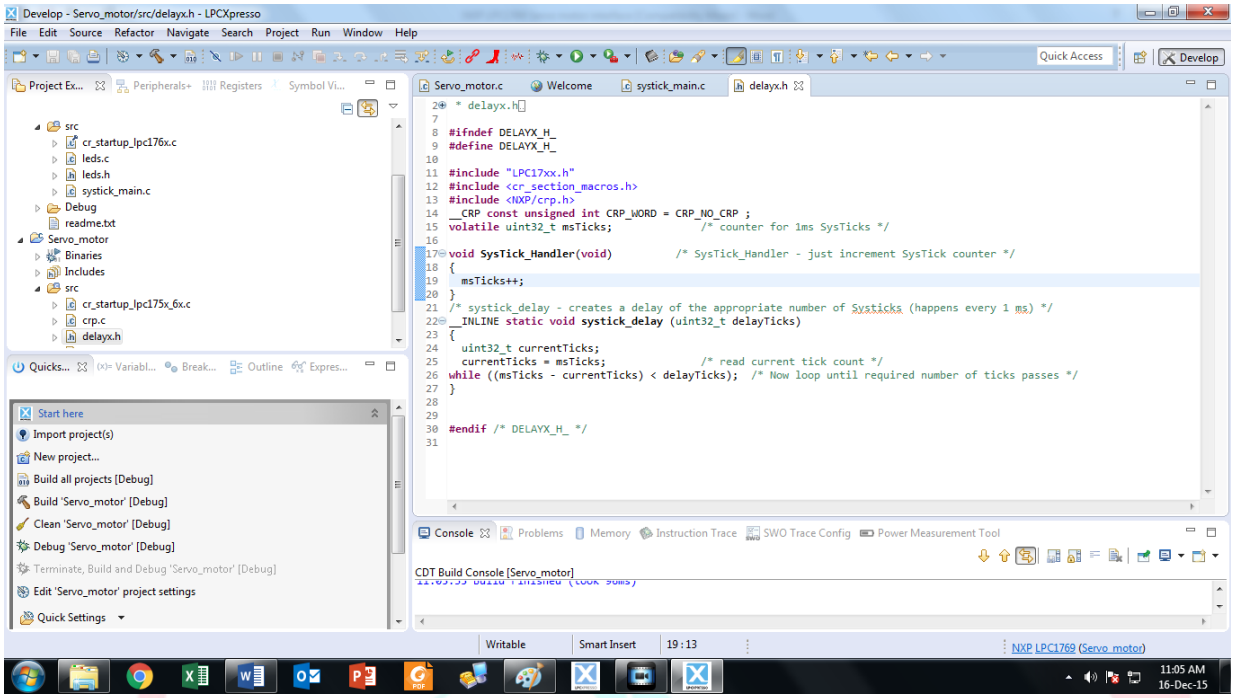


Figure 14

Step 15: Now we can start with our main program double clicking on Servo_motor.c.

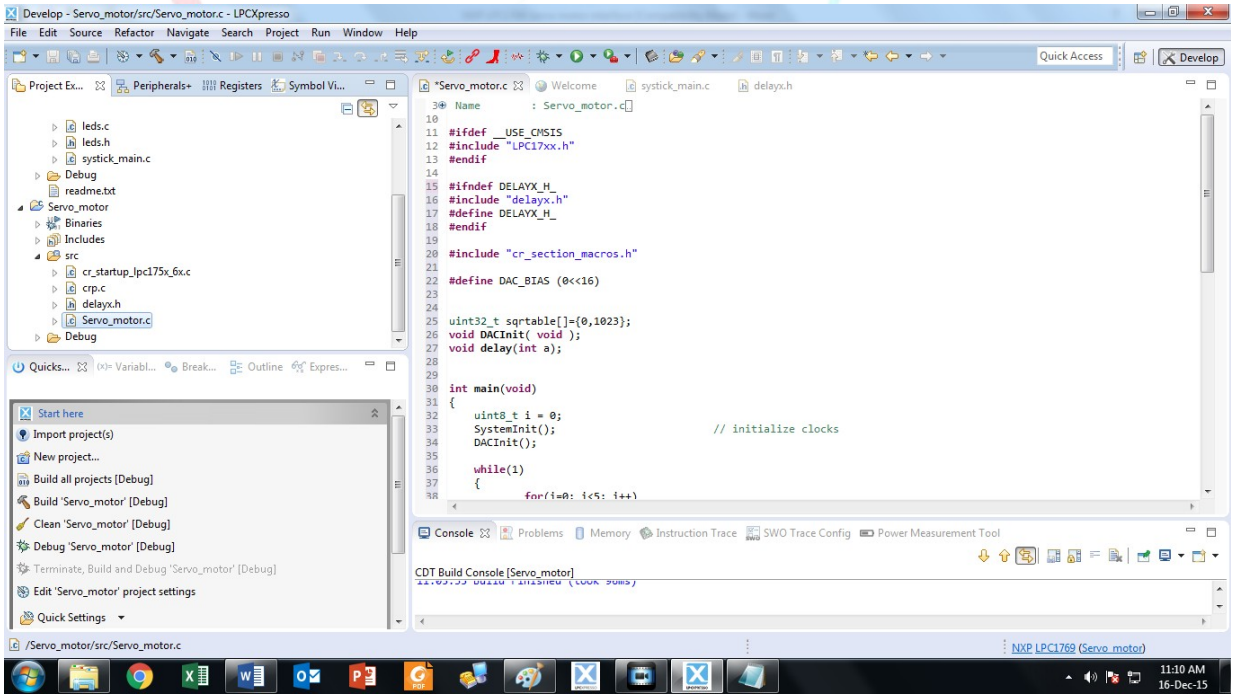


Figure 15

CODE:

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include "delayx.h"

#define DAC_BIAS (0<<16)

uint32_t pulse[]={0,1023};
void DACInit( void );
void delay(int a);


int main(void)
{
      uint8_t i = 0;
      SystemInit();              // initialize clocks
      DACInit();

    while(1)
      {
            for(i=0; i<5; i++)
            {
                  LPC_DAC->DACR = (pulse[1]<< 6)|DAC_BIAS;
                  delay (1);

                  LPC_DAC->DACR = (pulse[0]<< 6)|DAC_BIAS;
                  delay (25);
            }
            delay(1000);

            for(i=0; i<5; i++)
            {
                  LPC_DAC->DACR = (pulse[1]<< 6)|DAC_BIAS;
                  delay (2);

                  LPC_DAC->DACR = (pulse[0]<< 6)|DAC_BIAS;
                  delay (25);
            }
            delay(1000);
      }
}

void DACInit( void )
{
  LPC_PINCON->PINSEL1 &= ~0x00300000 ; //setup the related pin to DAC output
  LPC_PINCON->PINSEL1 |= 0x00200000;   //set p0.26 to DAC output - AOUT
  return;
}
```

```
void delay(int a)
{
	if (SysTick_Config(SystemCoreClock / 1000))
	{
	 while (1);
	 }
	systick_delay(a);
}
```

Step 16: After writing code, Build the project by clicking on Build Servo_motor on the Quickstart Panel on the bottom left of the window.

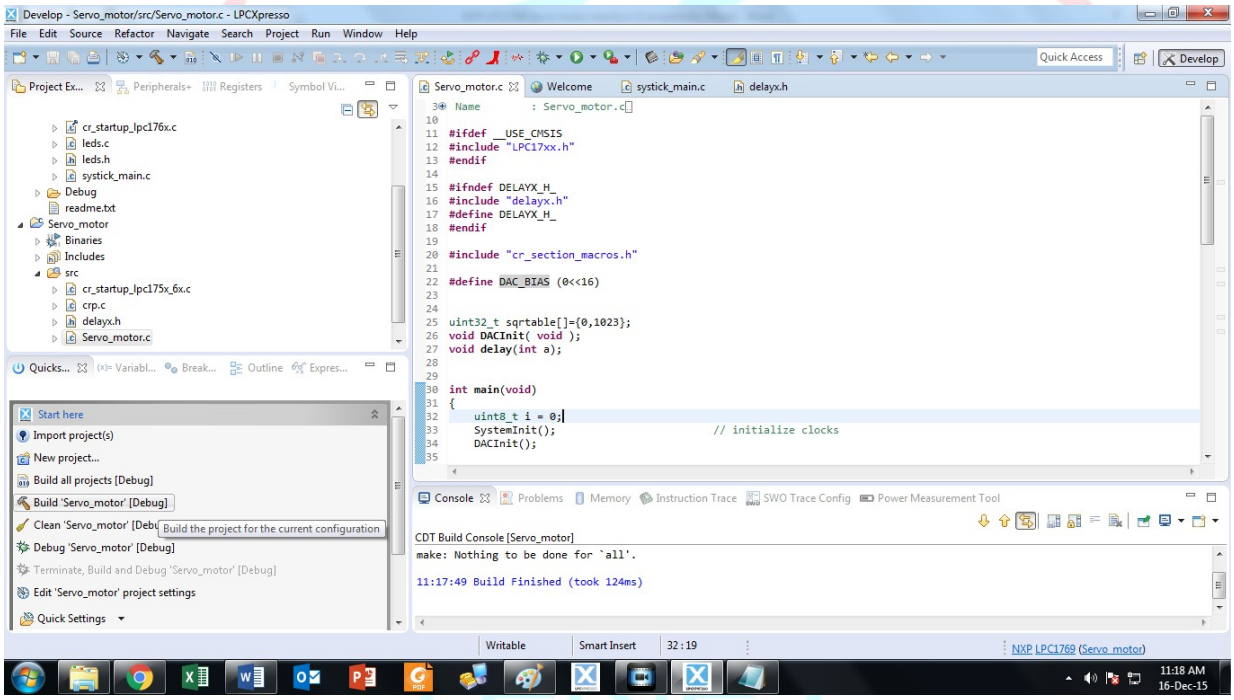


Figure 16

Step 17: Now, if all goes well connect the Micro B cable to LPC1769 and connect it to your computer. To upload the project file, click on the Program flash.

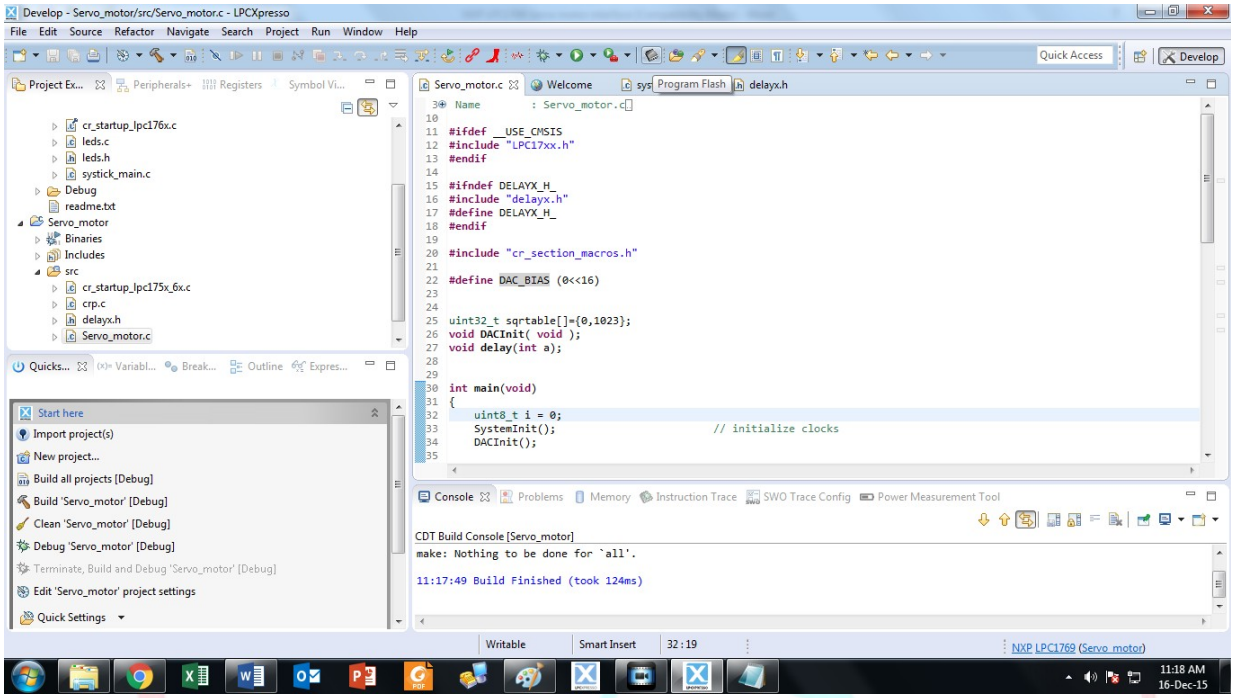


Figure 17

Step 18: Now select the Project file Servo_motor.axf. We can find it in our project folder.

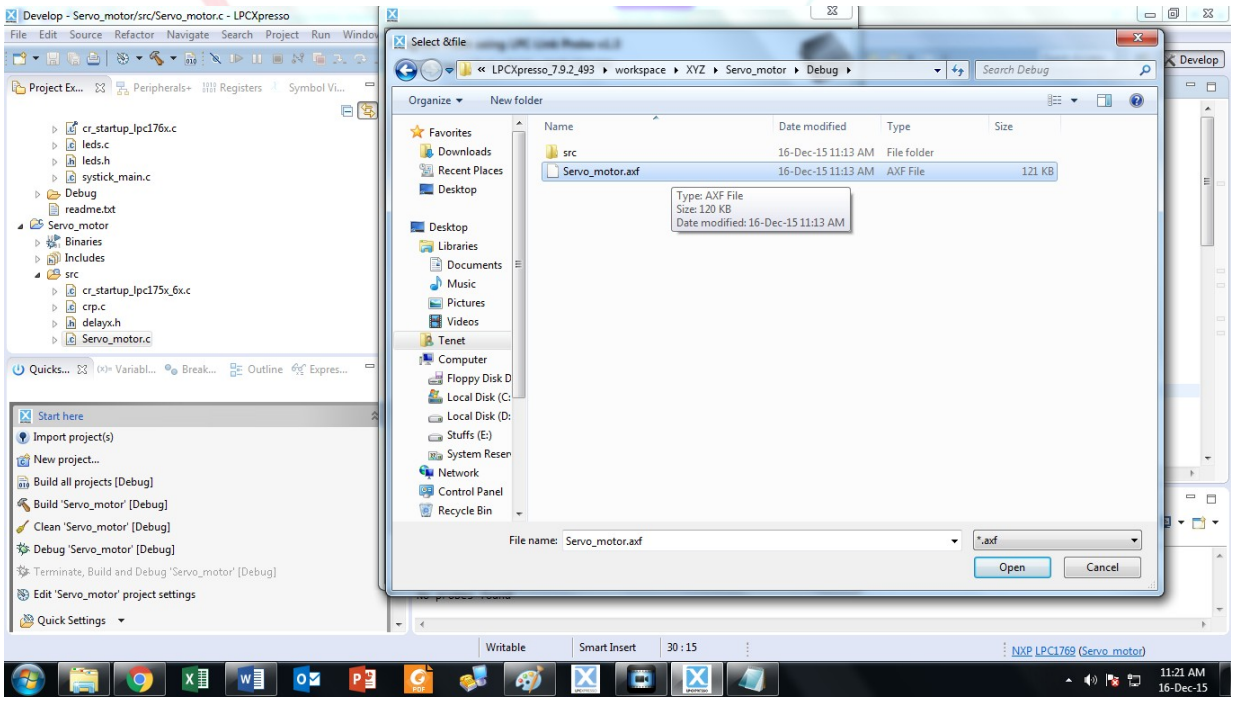


Figure 18

Step 19: Now this window shows we have finally dumped our project onto LPC1769.

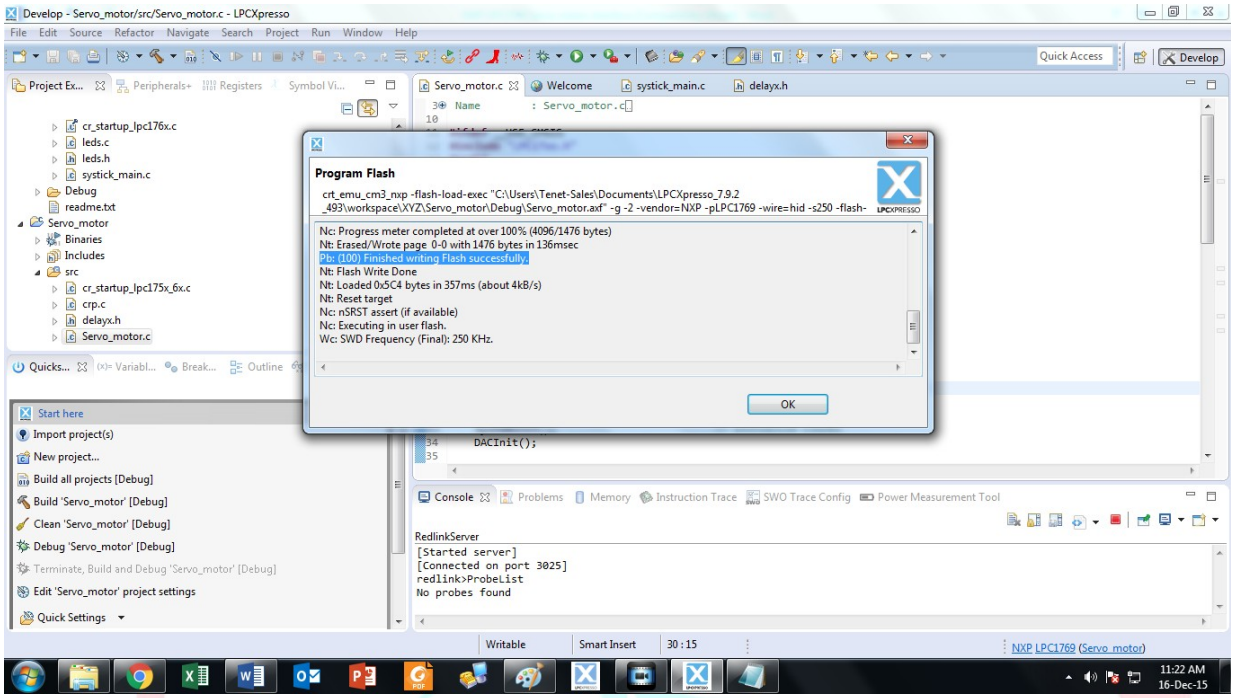


Figure 19

CIRCUIT EXPLANATION:

Hardware required:

- **LPC1769 Board:**
- **Servo motor:**
- **5V Power supply breakout:**
- **Jumper wires**

Servo motor:

Servo motor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration.

To understand how the servo motor works, let us know what's inside servo motor. Inside there is a pretty simple set-up: a small DC motor, potentiometer and a control circuit. As you can see in the picture below. The motor is attached by gears to the control wheel. As the motor rotates, the potentiometer's resistance changes, so the control circuit can precisely regulate how much movement there is and in which direction.

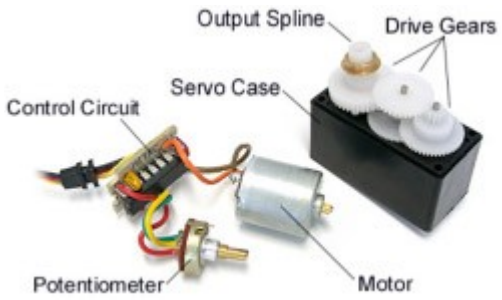


Figure 20

When the shaft of the motor is at the desired position, power supplied to the motor is stopped. If not, the motor is turned in the appropriate direction. The desired position is sent via electrical pulses through the signal wire. The motor's speed is proportional to the difference between its actual position and desired position. So if the motor is near the desired position, it will turn slowly, otherwise it will turn fast.

CONTROL PULSES:

How Servo motors are controlled? Servo motors are controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM), through the control wire. There is a minimum pulse, a maximum pulse and a repetition rate. A servo motor can usually only turn 90° in either direction for a total of 180° movement.

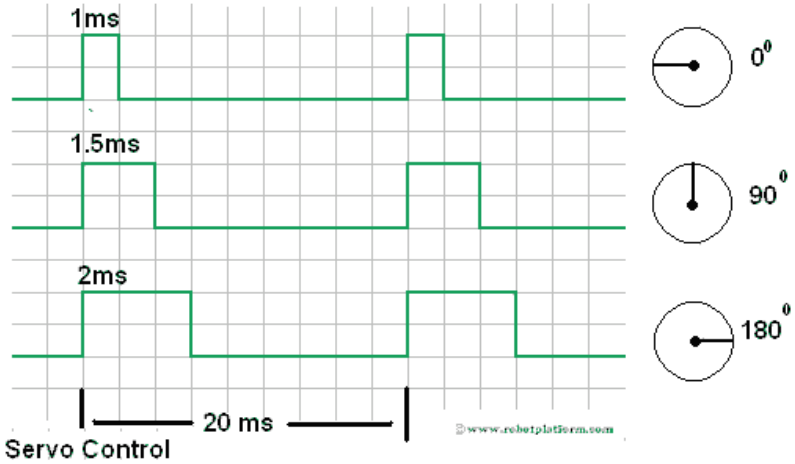


Figure 21

The PWM sent to the motor determines position of the shaft, and based on the duration of the pulse sent via the control wire the rotor will turn to the desired position. The servo motor expects to see a pulse every 20 milliseconds and the length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90° position. Shorter than 1.5ms moves it to 0° and any longer than 1.5ms will turn the servo to 180°.

CONNECTION DIAGRAM:

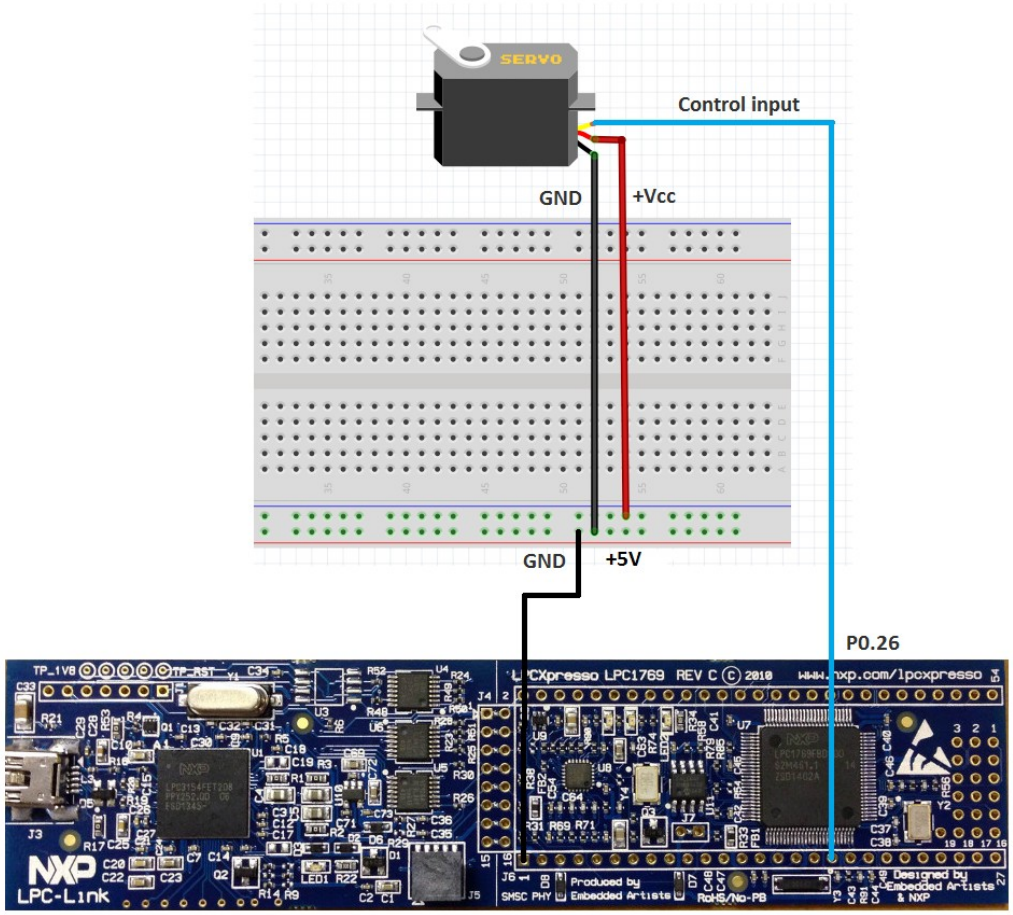


Figure 22

Note: In this application note we have used DAC instead of PWM. We can also use simple GPIO registers, just by producing high and low on a particular pin we can control servo motor. What matters is the precise delay between high and low signal. Best way to use is PWM peripheral of the micro-controller.

OUTPUT:

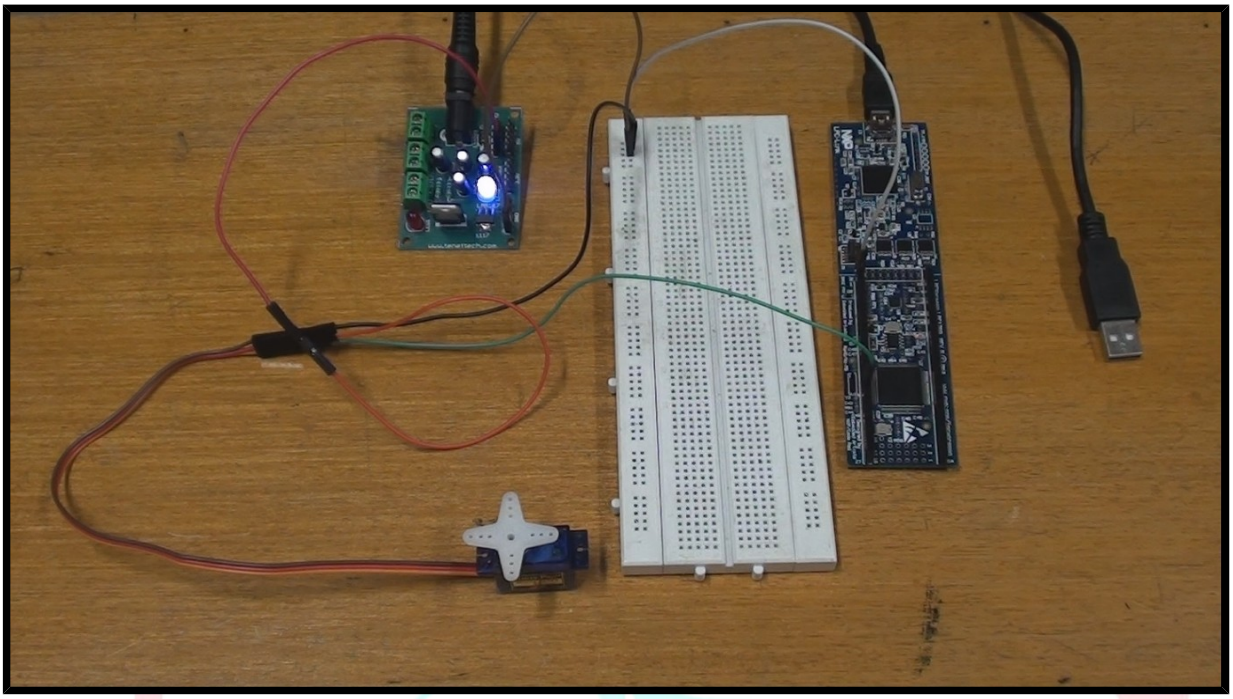


Figure 23

For product link:

1. **http://www.tenettech.com/product/1548/lpc1769-lpcxpresso-board**
2. **http://tenettech.com/product/6655/universal-gpio-board**
3. **http://www.tenettech.com/product/2846/power-supply-breakout**

For more information please visit: www.tenettech.com

For technical query please send an e-mail: info@tenettech.com