# MSP430 Microcontroller Setup and Workflow Documentation

This document outlines the steps taken to set up an MSP430 microcontroller development environment using MSP430-GCC and MSP430 Flasher tools. It covers the project structure, Makefile configuration, and how to use the setup both with and without an IDE (VS Code). This setup does not require Code Composer Studio (CCS) and provides a lightweight, portable workflow for embedded development.

## Step 1: Install the MSP430 Toolchain

I started by installing the necessary toolchain:  
- MSP430-GCC Toolchain (`msp430-elf-gcc`): Used for compiling C code for MSP430.  
- MSP430 Flasher Tool (`MSP430Flasher.exe`): Used to upload (flash) the compiled code to the microcontroller.  
These tools are typically installed in:  
- `C:/ti/msp430-gcc/` for the compiler.  
- `C:/ti/MSPFlasher\_1.3.20/` for the flasher tool.  
I’ve added these directories to your system PATH to ensure they can be accessed from any terminal.

## Step 2: Set Up Project Structure

My initial project directory structure was organized as follows:  
My-project/  
├── .vscode/ # VS Code configuration files (optional)  
├── makefile\_dir/  
│ └── Makefile # Makefile for build automation  
├── Src/  
│ ├── main.c # Main source file  
├── include/  
├── build/ # Directory for compiled files (output)  
```

## Step 3: Create Makefile for Build Automation

A Makefile was created to handle the build, link, conversion, and flashing process. Key targets in the Makefile:  
- `all`: Compiles and links the source files into `main.elf`.  
- `main.txt`: Converts `main.elf` to TI-TXT format.  
- `flash`: Uploads `main.txt` to the microcontroller using `MSP430Flasher.exe` with mspdebug reset.  
- `clean`: Removes all generated files.

## Step 4: Optional Integration with VS Code

To simplify the process in VS Code, `tasks.json` was created for integrating the Makefile commands:  
- \*\*Build Task\*\*: Runs `make all` to compile the project.  
- \*\*Clean Task\*\*: Runs `make clean` to delete generated files.  
- \*\*Flash Task\*\*: Runs `make flash` to upload the code to the microcontroller.  
This allows you to use VS Code's interface for building and flashing.

## Step 5: Verify Microcontroller Connection

The microcontroller connection was verified:  
- Checked that the MSP430 LaunchPad was detected by the system (Device Manager).  
- Identified the COM port (e.g., `COM3`), which is used for flashing.

## Step 6: Build and Flash Using Terminal

With the Makefile in place, the following commands can be used directly from the terminal without any IDE:  
- \*\*Build the Project\*\*:  
 ```bash  
 make -f makefile\_dir/Makefile all  
 ```  
- \*\*Convert to TI-TXT Format\*\*:  
 ```bash  
 make -f makefile\_dir/Makefile main.txt  
 ```  
- \*\*Flash the Microcontroller\*\*:  
 ```bash  
 make -f makefile\_dir/Makefile flash  
 ```  
- \*\*Clean the Project\*\*:  
 ```bash  
 make -f makefile\_dir/Makefile clean  
 ```

## Step 7: Test and Debug

A simple LED blink program was written and tested. If the LED did not blink initially, the program was manually started using `mspdebug`:  
```bash  
mspdebug ezfet  
> run  
```  
This confirmed that the code was uploaded and running on the microcontroller.

## Conclusion

This setup allows you to write, compile, and flash code for the MSP430 microcontroller directly from the terminal, without relying on Code Composer Studio (CCS). It is efficient, and portable. The integration with VS Code is optional, providing a more user-friendly interface if desired. This can also create more flexibility and modifications for any interface