

Lab 6

Keil uVision and Basic C Programming

Objectives:

- Create a new assembly project using Keil uVision5
- Learn the standard initialization steps of embedded software
- Perform basic C programming

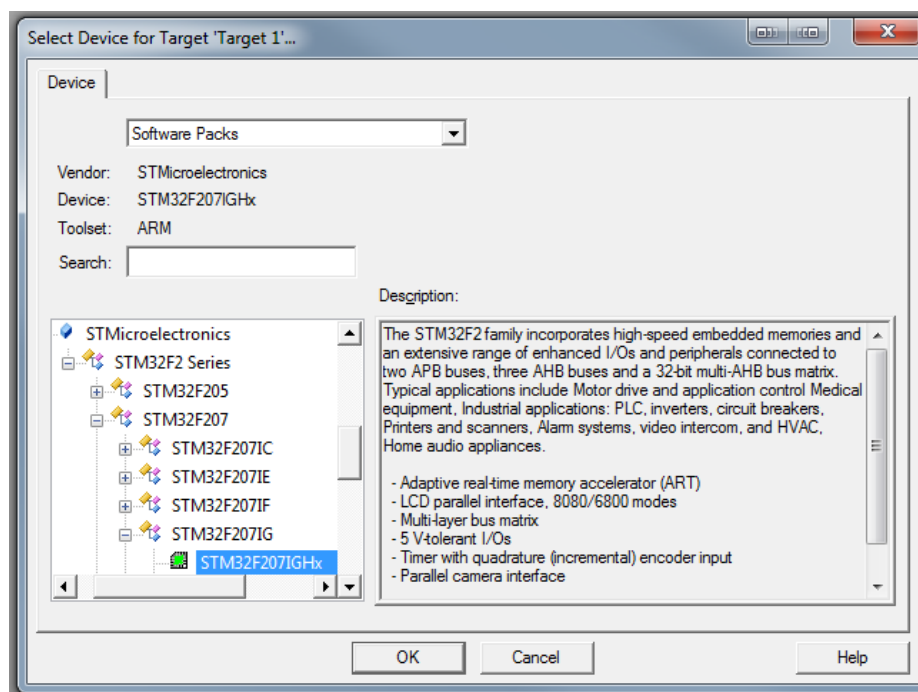
Files Needed:

- Lab6.c

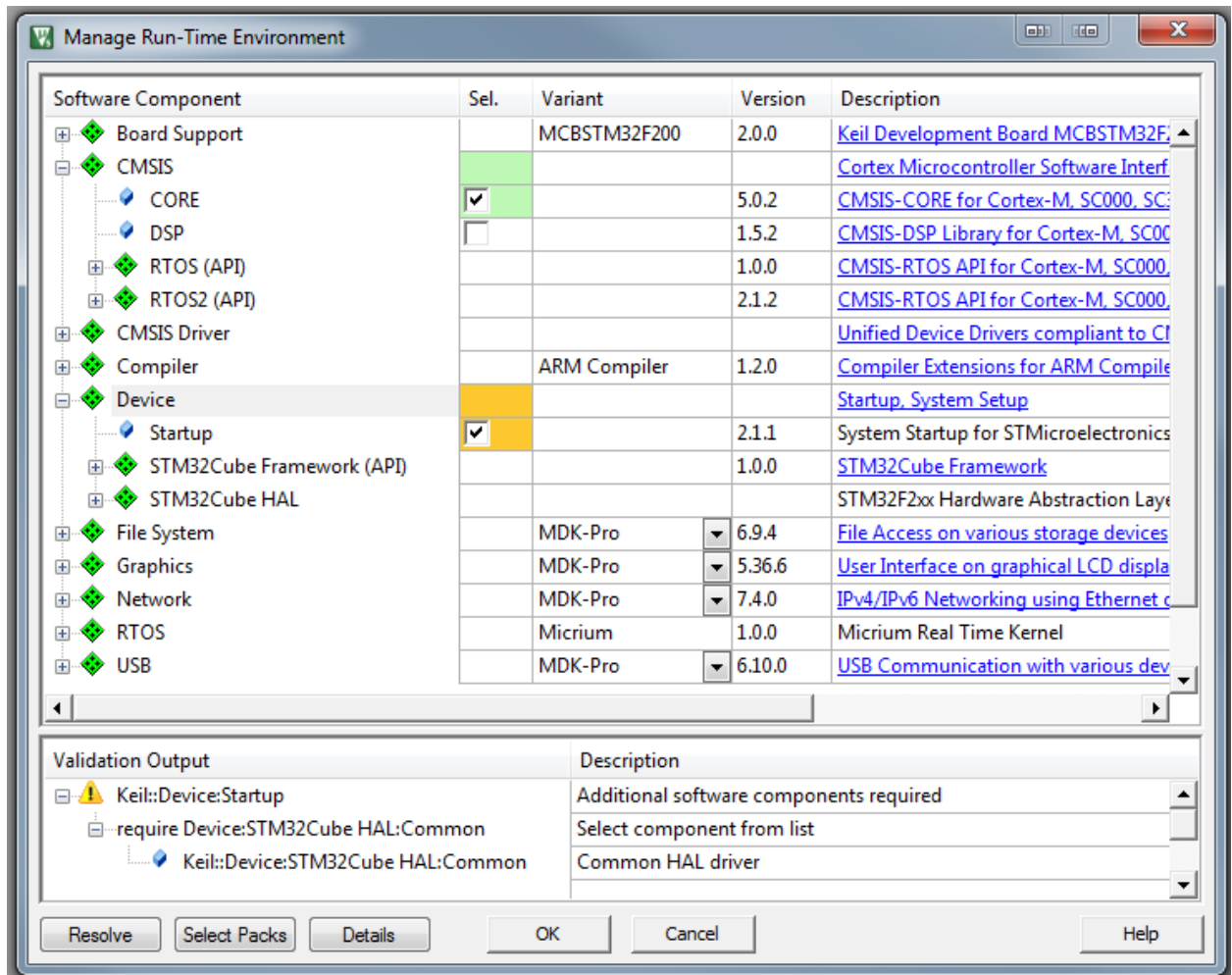
Assignment:

Create a new project using the following steps.

1. Open the Keil uVision5 development environment.
2. Click “Project”, then “New μ Vision Project...”
3. In the “Create New Project” dialog window, change to an appropriate directory and enter a name for the project file.
4. In the hierarchy, select “STMicroelectronics”, “STM32F2 Series”, “STM32F207IG”, “STM32F207IGHx”.

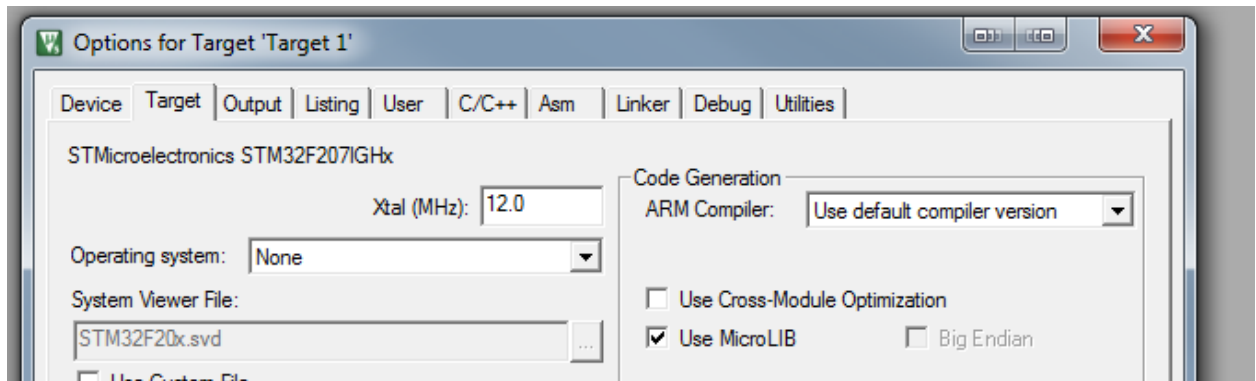


5. In the Manage Run-Time Environment dialog window, there are two components that (for now) always need to be added. Click on CMSIS->CORE, and click on Device-> Startup, then OK. Note that you can open this dialog again at any time to add and remove components from the project.

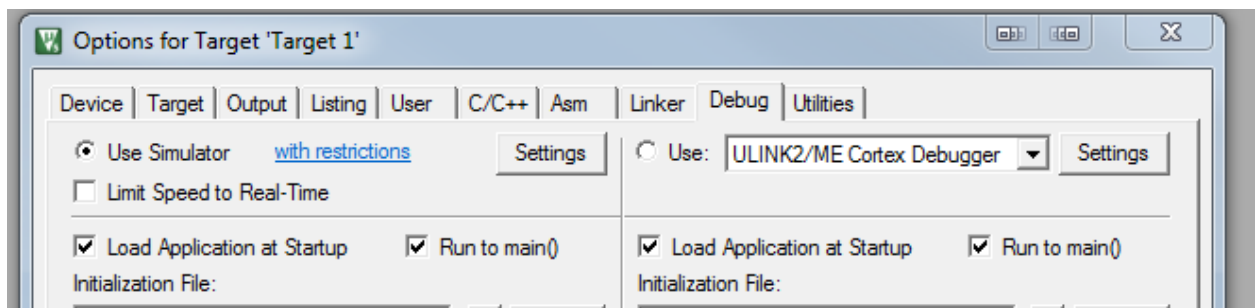




6. Copy the file lab6.c from Canvas to your project directory. This may be the same directory as the project file (*.uvprojx), or any subdirectory.
7. Add lab6.c to the project by right-clicking on "Source Group 1" and selecting "Add Existing Files to Group 'Source Group 1'...". The dialog box also allows you to keep adding files, so you will need to click "Add", then "Close".

8. Right click on “*Target1*” in the Project tab, click “*Options for Target ‘Target1’...*”, and click the “*Use MicroLIB*” checkbox in the “*Code Generation*” box of the “*Target 1*” tab.



9. Switch to the “*Debug*” tab in the Options window. When the Keil microcontroller board is being used, make sure that the “*ULINK2/ME Cortex Debugger*” is selected. For this lab, the software emulator can be used without needing the board, so make sure that “*Use Simulator*” is selected.



10. Build the project by clicking on the “*Build*” icon  in the upper left of the IDE window. The file should build without any errors or warnings.
11. Enter the debug mode by clicking the “*Start/Stop Debug Session*” icon . The IDE will warn you that only 32k of code is supported, which is fine.

The Debug IDE has several important windows:

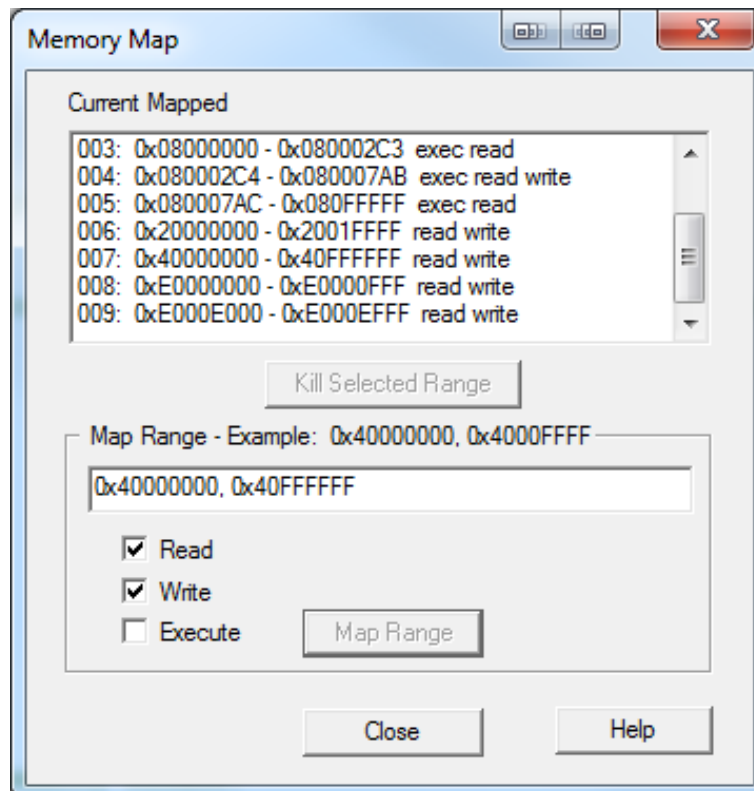
- Registers – These show the main 16 registers along with the program status register which can be expanded to show individual fields.
- Disassembly – This shows the address, machine code, and assembly code being executed, regardless of what format the source code was written in. The corresponding high level code, in this case the C code, is shown in dark red.

- Source code – This window (which has only tabs and no window name) shows the source files being executed. You can make changes to the files here without having to exit debug mode, but the changes won't take effect until the project has been rebuilt.
- Call Stack + Locals / Memory – monitor variables (once they are created in C) or monitor memory locations. To look at memory contents, you need to click on the *"Memory 1 tab"* and enter the starting location, such as 0x20000000.

The windows can be undocked and rearranged if desired. Feel free to explore the debug interface. Most buttons should be fairly straightforward. One handy note is that a breakpoint may be set by clicking in the grey left-hand margin in any code window, i.e. the disassembly, the source window, or the code window outside of the debugger.

There is an inconsistency when using the simulation mode that some of the memory addresses that are used during initialization of the processor are not in the simulation's memory map. You can handle this one of two ways. The first method is to just ignore it. When you run the program, you will step through about 10 lines of code that generate "error 65: access violation". These lines configure the onboard clock generator and have no effect on the simulation. Once you get to the main(void) function, your code will run correctly. This needs to be done every time you run the program.

The second approach is to tell the debugger about the missing range of memory. Click on "Debug", then "Memory Map...", and enter the range and options below, followed by clicking on "Map Range". This needs to be done every time you open the debugger.



Set a breakpoint at the “while(){}” loop on line 44 and run the program. It should stop at the breakpoint automatically. At this point, the function *filter()* has been called that creates a dummy output in memory. We need to know how to find it. There are two ways.

Method 1: Use a Watch window. If a Watch window isn’t already open (it probably isn’t by default), click on *View->Watch Windows-> Watch 1*. Then, in the *Name* column of the *Watch 1* window, type “*x1_n_coeff1*” without the quotes. Since uVision knows this is an array, you can click on the “+”, and it will expand the array into its values, as shown in the next figure.

Note that if you leave the debugger and then restart it, the Watch windows may be closed. However, if you reopen the Watch window, the variable you entered before will still be listed.

Watch 1			
Name	Value	Type	
x1_n_coeff1	0x200000AC x1_n_coeff1	unsigned in...	
[0]	0x00000001	unsigned int	
[1]	0x01010102	unsigned int	
[2]	0x02020203	unsigned int	
[3]	0x03030304	unsigned int	
[4]	0x04040405	unsigned int	
[5]	0x05050506	unsigned int	
[6]	0x06060607	unsigned int	
[7]	0x07070708	unsigned int	
[8]	0x08080809	unsigned int	
[9]	0x0909090A	unsigned int	
[10]	0x0A0A0A0B	unsigned int	
[11]	0x0B0B0B0C	unsigned int	
[12]	0x0C0C0C0D	unsigned int	
[13]	0x0D0D0D0E	unsigned int	
<Enter expression>			

Method 2: Open or select the Memory 1 window (it may initially be blank). This window shows the contents of the microprocessor in selectable formats. You may either type in a hexadecimal address or a variable name, and the window will show that range. In our case, type “x1_n_coeff1” without the quotes.

The Memory 1 window may be set to individual bytes by default. Since the array *x1_n_coeff1* contains 32-bit unsigned integers, right-click anywhere in the Memory 1 window, select “Unsigned”, then “Int”. This should rearrange the data as shown below. Your Memory 1 window may have a different number of columns and not all rows are shown, however the non-zero values should be in the same order.

Memory 1					
Address: x1_n_coeff1					
0x200000AC:	00000001	01010102	02020203	03030304	04040405
0x200000C0:	05050506	06060607	07070708	08080809	0909090A
0x200000D4:	0A0A0A0B	0B0B0B0C	0C0C0C0D	0D0D0D0E	00000000
0x200000E8:	00000000	00000000	00000000	00000000	00000000
0x200000FC:	00000000	00000000	00000000	00000000	00000000

You may have noticed that the hexadecimal number in the first row of the Memory 1 window, 0x200000AC for this case, matches the value of the array variable in the Watch 1 window. This is the address in memory where the array starts. If this address is known, it may be typed directly into the space at the top of the Memory Window.

Lab 6 has several places with “LAB 6” comments indicating where you need to add code to modify the file. Modify the file so that each of the 3 sets of coefficients are used on each of the 2 input sequences with the filter function. You will need to write the filter function to solve the differential equations based on the coefficients and input sequence passed (similar to what was done manually in a homework assignment and on the first midterm). The filter only uses current and previous values of the input. It does not use previous values of the output. For a filter with N coefficients, assume the initial conditions for the output for output[0] to output[N-1] are 0.

When your program works, submit the code and a clear screen clipping of just the memory window for each of the 6 outputs (i.e. start the memory window at the output for each case, and don't just give one large clipping with all 6 back-to-back). Please make sure to label each of the 6 figures in the PDF report.

As a helpful hint, pressing Shift+Windows+s allows you to easily take a screen clipping.

Deliverables:

Upload an electronic copy of the deliverables to Canvas per the lab submission guidelines. Please note: **ALL** deliverables must be present in the single PDF report.

- Lab6.c file with additional code.
- Screenshots of output arrays in Memory tab showing each of the 6 output arrays.