## Optimize Large FIR Energy Calculation

### Purpose

To demonstrate some basic optimization techniques

This lab requires a SINGLE EVM board (EVM6678 or EVM6670). The lab can be performed using the estimated cycle count 6678 simulator or 6670 simulator

### Project Files

The following files are used in this lab:

1. firMain.c
2. naturalCFilter.c
3. intrinsicCFilter.c
4. utilities.c
5. test.h
6. linkerNoRTSC\_L2.cmd

### TASK 1: Build and run the Project

1. Open CCS
2. Create new project
3. Create new project file location
4. Delete default main.c
5. Copy project files to new location.
6. Look at the source code. Understand what the main code is doing.
7. Open Properties and set the debug model to full and optimization levels to minimum.
8. Build the project. If there is an open file error add the path to include file (next step)
9. Set the include path to point to the PDK.
10. Start a debug project. Compile and link. Generate an out file
11. If the target is not defined yet, define a target. Assign gel file to core 0.
12. Launch the target debugger (emulation), connect and load the program into core 0
13. From the run menu, enable the clock
14. Run the code and record the cycles time for natural C function and for intrinsic function

### TASK 2: Optimization

1. Change the project build option. Suppress all debug features and enable the highest time optimization.
2. Re-build and re-run. Record the optimized project cycles time for natural C function and for intrinsic function

Challenge Question? How much improvement you notice for the natural C code and for the intrinsic code?

Is there something wrong with the code?

1. Intrinsic functions utilized the processor better. They should give better results

### TASK 3: Software pipeline

1. Keep the assembly file. In the project properties build compiler tab go to assembly option and check the appropriate tab
2. Rebuild the code. Find the assembly file. Hint, if the build configuration is debug, the assembly files are in the debug directory.
3. Open the interisicCFilter.asm file. Was the compiler able to find software pipeline?

Challenge Question? What are the reasons in general that the compiler cannot schedule software pipeline?

1. Think about cases that can cause randomness in the execution timing
3. Open the interisicCFilter.asm file and find the reason why the compiler could not schedule software pipeline

Challenge Question? What is the reason that the compiler cannot schedule software pipeline in that case? How to resolve it

1. Think about inline function
3. Substitute the intrinsic function instead of the regular function in all the loops
4. Re-build and re-run. Look at the intrinsicCFilter.asm. Did the compiler schedule software pipeline?
5. Record the optimized project cycles time for natural C function and for intrinsic function with software pipeline
6. Enable the pragma must iterate everywhere in the code
7. Re-build and re-run. Look at the intrinsicCFilter.asm. Compare with previous asm file
8. Record the optimized project cycles time for natural C function and for intrinsic function with software pipeline and must iterate pragma

### TASK 4: Aligned Data

1. In the intrinsicCFilter.c code, the data is read from the memory.

Challenge Question? What is the alignment of the input data? What is the alignment of the filter coefficients (in the stack)?

1. Find Pragma that align the data. What other ways there to align the data on 64 bit boundary?
2. Change the code to tell the compiler that the data is loaded from aligned memory
3. Re-build and re-run
4. Record the optimized project cycles time for natural C function and for intrinsic function with software pipeline and aligned load.

### TASK 5: Cache Considerations

1. In test.h, change the number of elements to 4K, 8K and 16K
2. Record the cycle counts for each case

Challenge Question? Why is the non-linear jump in the performances

1. Think about cache trashing
2. Change the code to take full advantage to the cache
3. Break the data into chanks, and call each routine multiple times. Make sure to keep the sum between calls
4. Re-build and re-run
5. Record the final optimization cycle count.
6. Any ideas how to speed to code further?

*Maintain Blank Page Here*