**Format for Intermediate Reports on DSP Optimizations**

**Group Name** : Speech group 5

**Team member 1** : Koen Goetshalckx

**Team member 2** : Seppe Iven

Selfie(s). Illustrating who is who.

Seppe: Koen:

 

# Report 1

**Lab date : 22/04, 2016**

**Submission time : 23/04**

## Logs:

We have found and fixed some bugs that occurred when implementing another group’s crypto part. We installed and configured CCS 5 and went through the given slides. We made the code slightly more modular. We profiled the code running on the DSP-simulator for a first time. We tried to speed up convolve, but no significant change was realized (see discussion below).

## Profile results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Major Optimization** | **Functions Effected** | **Total Cycles\*** | | **Change (in %)** |
| **Before** | **After** |
| Base Code |  |  | 21,8M |  |  |
| Session 1 | Hardcoded filter in unrolled inner loop of convolve for filter2 and filter3 | Convolve 🡪 convolveFilter2Odd, convolveFilter2Even, convolveFilter3Odd, convolveFilter3Even | 21,8M | 21,7M | 0,5 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Function Name | File Name | Total no. of calls | Exclusive Count Total\* | | | Inclusive Count Total\* | | | Optimization carried out |
| Before  (Previous Session) | After  (This Session) | % Change | Before  (Previous Session) | After  (This Session) | % Change |
| Convolve  🡪  Convolve,  convolveFilter2Odd, convolveFilter2Even, convolveFilter3Odd, convolveFilter3Even | Analysis.c | 3840 | 3,5M | 3,4M | 3 | 13,5M | 13,4M | 1 | Hardcoded filter coefficients in an unrolled inner loop |

## Discussion (important)

The inner loop of the convolve function is the most used code of the program. Before the DSP, this inner loop was already sped up by a factor of 2, using the profiler of Visual Studio and the default windows C compiler. Now, an attempt to speed it up even more was made by unrolling this inner loop and hardcoding the filter coefficients in it. Some noticeable gain was expected, considering unrolled code improves pipelining and the hardcoded coefficients should require less data accesses. However, only a very small gain in speed was realized. Therefore, and because of the unreadability of the manually unrolled code (manually to hardcode the coefficients), the changes were reverted. (Only filter2 and filter3 were hardcoded. This should be enough to test the impact of the change.)

The amount of cycles is for an input file of 2986 samples

# Report 2

Lab date: 28/4

Submission date: 29/4

Submission deadline: 29/4

## Logs:

Changed some shorts to chars (lengths and offset). This changed the total amount of cycles to 21.6M.   
Declared a lot of variables ‘near’, but this gave no noticeable change, so we reverted it for cleanliness.  
Used the keyword ‘const’ for variables that behave like constants. This gave almost no speed up at all, so we reverted it for cleanliness.  
Combined left and right in convolve -> less dependencies & overhead

## Profile results:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Function Name | File Name | Total no. of calls | Exclusive Count Total\* | | | Inclusive Count Total\* | | | Optimization carried out |
| Before  (Previous Session) | After  (This Session) | % Change | Before  (Previous Session) | After  (This Session) | % Change |
| Convolve | Analysis.c | 1920 | 3.4M | 2.2M | 35% | 13.4M | 12.0M | 10% | Combine |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Major Optimization** | **Functions Effected** | **Total Cycles\*** | | **Change (in %)** |
| **Before** | **After** |
| Base Code |  |  | 21,8M |  |  |
| Session 1 | Hardcoded filter in unrolled inner loop of convolve for filter2 and filter3 | Convolve 🡪 convolveFilter2Odd, convolveFilter2Even, convolveFilter3Odd, convolveFilter3Even | 21,8M | 21,7M  REVERTED | 0,5 |
| Session 2 | Combine convolve | Convolve | 21,8M | 20.1M | 8% |

## Discussion (important)

As expected, combining left and right convolve nets a gain in cycles because both the dependencies and the overhead are reduced. A lot of the work for the left and right convolve (e.g. fetching filter coefficients) needs to be done only once. The compiler has more independent variables to use for parallelism/pipelining. A slightly larger gain was expected, however.  
Most of the small changes, such as declaring variables ‘near’ and ‘const’ are reverted because they gave almost no gain. It is not excluded that these changes may be implemented again in the future.

# Report 3

Lab date: 29/4

Submission date: 30/4

Submission deadline: 30/4

## Logs:

Changed type of ‘result’ in the inner loop of convolve to int.

Spent most of our time fixing a bug in the rewritten version of convolve (see below)

## Profile results:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Function Name | File Name | Total no. of calls | Exclusive Count Total\* |  |  | Inclusive Count Total\* |  |  | Optimization carried out |
|  |  |  | Before  (Previous Session) | After  (This Session) | % Change | Before  (Previous Session) | After  (This Session) | % Change |  |
| Convolve | Analysis.c | 1920 | 2.2M | 1.1M | 50% | 12.0M | 2.0M | 83% | Long long to int |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Major Optimization** | **Functions Effected** | **Total Cycles\*** |  | **Change (in %)** |
|  |  |  | **Before** | **After** |  |
| Base Code |  |  | 21,8M |  |  |
| Session 1 | Hardcoded filter in unrolled inner loop of convolve for filter2 and filter3 | Convolve convolveFilter2Odd, convolveFilter2Even, convolveFilter3Odd, convolveFilter3Even | 21,8M | 21,7M  REVERTED | 0,5 |
| Session 2 | Combine convolve | Convolve | 21,8M | 20.1M | 8% |
| Session 3 | Long long to int | Convolve | 20.1M | 10.2M | 49% |

## Discussion (important)

We rewrote convolve: instead of doing the wrapping around to the beginning of an array in the inner loop, we now temporarily copy the array, with the copy being in the right order. This turns the inner loop into a classic multiply and add loop, which the compiler easily optimizes, at the expense of a small increase in memory usage. This only gave a small gain though.  
A much larger gain was realized by changing the intermediate result in the inner loop to an int, instead of a long long. This change may theoretically cause overflows, but implementing this exactly the same in MATLAB (our C and MATLAB code produce the exact same solution down to every bit) showed that the output signals remain unchanged.