SENG 515 Group 12 Final Project Report

By: Ryan Lau, Alex On, Julien Rabino, Matthew McDougall

Part 1:

Functionality

This section of the project consists of four principal functions which process digital samples for a filtering application. Variations in the block size, memory model, and the use of unrolling via MLA were applied across each function. Below is a description of the inner workings of each function:

```
static int16_t ProcessSampleCircular(int16_t newsample, int16_t* history) {

    // set the new sample as the head
    history[head] = newsample;

    // set up and do our convolution
    int tap = 0; // indexing filter_coeffs
    int current = head; //  indexing history array
    int32_t accumulator = 0;
    for (tap = 0, current = head; tap < NUMBER_OF_TAPS; tap++, current--) {
        current = (current + NUMBER_OF_TAPS) % NUMBER_OF_TAPS;
        accumulator += (int32_t)filter_coeffs[tap] * (int32_t)history[current];
        }

    // assign new head and tail (if buffer full)
    head = (head + 1) % NUMBER_OF_TAPS;



    if (accumulator > 0x3FFFFFFF) {
        accumulator = 0x3FFFFFFF;
        overflow_count++;
    } else if (accumulator < -0x40000000) {
        accumulator = -0x40000000;
        underflow_count++;
    }
```

```
    int16_t temp;
    if (accumulator < 1000 && accumulator > -1000) {
        temp = (int16_t)accumulator;
    } else{
        temp = (int16_t)(accumulator >> 15);
    }
    return temp;
}
```

This function implements the simplest filter using a circular buffer, processing one sample at a time. It stores the new sample as the head of the buffer, then prepares the buffer for and performs convolution using the filter coefficients. The buffer is then modified to accommodate the next sample, using the modulus operator since it forms a circular buffer. Overflow and underflow checks are then applied, and necessary modifications are made. Finally, the accumulator value is formatted for fixed representation and returned.

```
static int16_t ProcessSampleCircular2(int16_t newsample, int16_t* history) {

    // set the new sample as the head
    history[head] = newsample;

    // set up and do our convolution
    int tap = 0; // indexing filter_coeffs
    int current = head; // indexing history array
    int32_t accumulator = 0;

    // unrolling the loop to process two taps at a time
    for (tap = 0; tap < (NUMBER_OF_TAPS / 2) * 2; tap += 2) {

        current = (head - tap) % NUMBER_OF_TAPS;
        int history_idx_1 = (current + NUMBER_OF_TAPS - 1) % NUMBER_OF_TAPS;
        int history_idx_2 = (current + NUMBER_OF_TAPS) % NUMBER_OF_TAPS;

        // first tap (for tap)
        __asm volatile ("SMLABB %[result1], %[op1], %[op2], %[acc1]"
                        : [result1] "=r" (accumulator)
                        : [op1] "r" ((int32_t)filter_coeffs[tap]),
                          [op2] "r" ((int32_t)history[history_idx_2]),
                          [acc1] "r" (accumulator)
                        );
```

```
        // second tap (for tap+1)
        __asm volatile ("SMLABB %[result2], %[op1], %[op2], %[acc2]"
                        : [result2] "=r" (accumulator)
                        : [op1] "r" ((int32_t)filter_coeffs[tap + 1]),
                          [op2] "r" ((int32_t)history[history_idx_1]),
                          [acc2] "r" (accumulator)
                        );
    }

    // if NUMBER_OF_TAPS is odd, process the last tap
    if (NUMBER_OF_TAPS % 2 != 0) {
        current = (current + (NUMBER_OF_TAPS - 2)) % NUMBER_OF_TAPS;
        __asm volatile ("SMLABB %[result], %[op1], %[op2], %[acc]"
                        : [result] "=r" (accumulator)
                        : [op1] "r" ((int32_t)filter_coeffs[NUMBER_OF_TAPS - 1]),
                          [op2] "r" ((int32_t)history[current]),
                          [acc] "r" (accumulator)
                        );
    }

    // assign new head and tail (if buffer full)
    head = (head + 1) % NUMBER_OF_TAPS;

    // handle overflow/underflow
    if (accumulator > 0x3FFFFFFF) {
        accumulator = 0x3FFFFFFF;
        overflow_count++;
    } else if (accumulator < -0x40000000) {
        accumulator = -0x40000000;
        underflow_count++;
    }

    // convert accumulator to int16_t
    int16_t temp;
    if (accumulator < 1000 && accumulator > -1000) {
        temp = (int16_t)accumulator;
    } else {
        temp = (int16_t)(accumulator >> 15);
    }

    return temp;
}
```

This function follows a similar structure to the previous filter function, mostly due to the fact that it also implements a circular buffer. The key difference applied in this function is that loop unrolling is applied, meaning that (in this case) two samples will be processed in each loop iteration. This is a common optimization which can be scaled further than two samples. It can, however, introduce unsafe memory practices. One of these effects is seen in the above function, where an odd-sized number of taps will cause invalid memory accesses to occur. This fault is avoided by introducing a check for odd-sized coefficient arrays.

```
static int  ProcessBlock(int16_t newsample, int16_t* history) {

    history[head] = newsample;
    if (samples_since_last_frame++ < FRAME_SIZE - 1 ){
        head = (head + 1) % HISTORY_SIZE;

        return false;
        // returning false means that there are not enough samples to for the frame to be processed
    }
    samples_since_last_frame = 0;


    // processing the frame
    int tap, current;
    int32_t accumulators [FRAME_SIZE] = {0}; // accumulator[2] corresponds to the newest sample
    for (tap = 0, current = head; tap < NUMBER_OF_TAPS; tap++, current--) {
            int32_t coeff = (int32_t)filter_coeffs[tap]; // loading the coefficient once

            int base_idx = (current - MAX_FRAME_IDX + HISTORY_SIZE) % HISTORY_SIZE;

            for (int i = 0; i < FRAME_SIZE; i++){
                int history_idx = (base_idx + i) % HISTORY_SIZE;
                accumulators[i] += coeff * (int32_t)history[history_idx];
            }
    }


    head = (head + 1) % HISTORY_SIZE; // moving the head


    for (int i = 0; i < FRAME_SIZE; i++){
        if (accumulators[i] > 0x3FFFFFFF) {
                accumulators[i]= 0x3FFFFFFF;
                overflow_count++;
            } else if (accumulators[i] < -0x40000000) {
```

```
                accumulators[i] = -0x40000000;
                underflow_count++;
            }

        if (accumulators[i] < 1000 && accumulators[i] > -1000) {
            accumulators_16[i] = (int16_t)accumulators[i];
        } else{
            accumulators_16[i]= (int16_t)(accumulators[i] >> 15);
        }
    }
    return true;
}
```

This function takes a different approach than the previous filtering functions by implementing a frame-based processing filter (although it technically does also use a circular buffer, it operates in a largely different way). The same critical components are present, namely the final return statements and the overflow checks. The main difference in this function is the usage of a frame to relegate data accesses and processing to occur less frequently and all at once. This is shown in the early return statement at the beginning of the function (where most samples will return). Although this strategy can be more efficient, oftentimes sample-based processing will be preferred for hard real-time requirements.

```
static int ProcessBlock2(int16_t newsample, int16_t* history) {
    // Set the new sample as the head
    history[head] = newsample;

    if (samples_since_last_frame++ < FRAME_SIZE - 1) {
        head = (head + 1) % HISTORY_SIZE;
        return false;
    }

    samples_since_last_frame = 0;

    // Processing the frame
    int current = head;
    int32_t accumulators[FRAME_SIZE] = {0};  // accumulator[2] corresponds to the newest sample
    int tap;

    // Unrolling the outer loop to process multiple taps at once
    for (tap = 0; tap < (NUMBER_OF_TAPS / 2) * 2; tap += 2, current-=2) {
        int32_t coeff1 = (int32_t)filter_coeffs[tap];     // First tap coefficient
        int32_t coeff2 = (int32_t)filter_coeffs[tap + 1]; // Second tap coefficient
```

```
    current = (current + HISTORY_SIZE) % HISTORY_SIZE;

    int base_idx = (current - FRAME_SIZE + HISTORY_SIZE) % HISTORY_SIZE;

    for (int i = 0; i < FRAME_SIZE; i++) {
        int history_idx_1 = (base_idx + i) % HISTORY_SIZE;
        int history_idx_2 = (base_idx + i + 1) % HISTORY_SIZE;

        // Separate the assembly instructions for each tap
        __asm volatile(
            "SMLABB %[result1], %[op1], %[op2], %[acc1];"
            : [result1] "=r" (accumulators[i])
            : [op1] "r" (coeff1),
              [op2] "r" ((int32_t)history[history_idx_2]),
              [acc1] "r" (accumulators[i])
        );

        __asm volatile(
            "SMLABB %[result2], %[op1], %[op2], %[acc2];"
            : [result2] "=r" (accumulators[i])
            : [op1] "r" (coeff2),
              [op2] "r" ((int32_t)history[history_idx_1]),
              [acc2] "r" (accumulators[i])
        );
    }
}

// If NUMBER_OF_TAPS is odd, process the last tap
if (NUMBER_OF_TAPS % 2 != 0) {
    int32_t coeff = (int32_t)filter_coeffs[NUMBER_OF_TAPS - 1];
    int base_idx = (current - MAX_FRAME_IDX + HISTORY_SIZE) % HISTORY_SIZE;

    for (int i = 0; i < FRAME_SIZE; i++) {
        int history_idx = (base_idx + i) % HISTORY_SIZE;

        // Accumulate for the last tap
        __asm volatile(
            "SMLABB %[result], %[op1], %[op2], %[acc];"
            : [result] "=r" (accumulators[i])
            : [op1] "r" (coeff),
              [op2] "r" ((int32_t)history[history_idx]),
              [acc] "r" (accumulators[i])
        );
    }
}
```

```
    // Moving the head to the next position
    head = (head + 1) % HISTORY_SIZE;

    // Handling overflow/underflow and updating the accumulator values
    for (int i = 0; i < FRAME_SIZE; i++) {
        if (accumulators[i] > 0x3FFFFFFF) {
            accumulators[i] = 0x3FFFFFFF;
            overflow_count++;
        } else if (accumulators[i] < -0x40000000) {
            accumulators[i] = -0x40000000;
            underflow_count++;
        }

        if (accumulators[i] < 1000 && accumulators[i] > -1000) {
            accumulators_16[i] = (int16_t)accumulators[i];
        } else {
            accumulators_16[i] = (int16_t)(accumulators[i] >> 15);
        }
    }

    return true;
}
```

This function builds on the previous function by maintaining the same frame-based processing while unrolling the loop as in ProcessCircular2().

Assembly

Using the STMCube IDE, we generated assembly code for analysis in this report, which can be found in the appendix of this document. Below are some observations regarding this assembly code:

- The most obvious difference across each function is the difference in total instruction count from the circular buffer to the frame-based processing, which can be seen in the program sizes of the experimental data
- Jump and branch instructions occur less frequently in the unrolled-loop functions relative to their total instruction count, as more instructions are present in total. Because loop unrolling reduces the total number of iterations a loop will run, it is also known that these jump and branch instructions are executed less often
- A great number of memory-related instructions can be found in the disassembly for the frame-based processing functions, especially in those compiled with a larger block size (of 16)

- As with memory-related instructions, the frame-based processing functions, especially those which have unrolled loops, contain swaths of arithmetic instructions which handle parallel computations

Experiment

Function	Block Size	Cycles Per Sample – Size of Program (KB)					
		-O0		-Os		-Ofast	
ProcessCircular()	-	12210	27.70	3639	23.76	3638	24.40
ProcessCircular2()	-	11300	24.69	2736	24.34	2729	24.69
ProcessBlock()	3	19100	28.01	5500	23.93	2953	24.68
ProcessBlock()	16	15548	28.05	5516	23.98	4012	27.59
ProcessBlock2()	3	16694	28.14	6133	24.02	2733	24.73
ProcessBlock2()	16	13939	28.17	4392	24.06	2028	27.39

Most of the anticipated trends are true according to this data. We see that:

- Size optimization always significantly decreased program size and always resulted in the smallest program size
- Speed optimization always required the fewest number of cycles per sample, especially for frame-based processing
- In general, program size was similar across functions using unrolled loops and their counterparts
- In general, increasing block size resulted in a reduction in the value of cycles per second - an exception for this trend is the optimized and non-unrolled-loop function ProcessBlock(), which saw no significant change when increasing block size
- In optimized builds, block size had a significant impact on the effectiveness of loop unrolling. In particular, the very small and odd-numbered block size of 3 was difficult to handle alongside the loop unrolling by a factor of 2 for the compiler when using optimizations, and resulted in a performance **drop**

Key takeaways from this data suggest that optimized code, whether manual or using compiler flags, is massively important for the performance of the program. It is also useful in reducing program size and trimming bloat. Interestingly, some hiccups occur when attempting to combine manual and automatic optimization techniques, as the compiler tends to see any manual optimization as a functional requirement, i.e. any code that is to be optimized by the compiler is taken at face value and is essentially a restriction. In general, we can also see that frame-based processing should be utilized with large frame sizes (much larger than those used in this project) as we can predict their performance would continue to increase with frame size. This further emphasizes that frame-based processing should be employed for low-latency tasks which do not have hard real-time constraints in order to take advantage of the potential performance benefits.

Considering an 8kHz sampling rate, we can calculate which functions can satisfy the timing requirements. First, we find the maximum time to process one sample:

$$\frac{1}{8kHz} = 0.125ms$$

Then, we can find the time to process one sample for any function using its value for cycles per second, with the processing rate of the chip:

$$\frac{6133}{100MHz} = 0.061ms$$

We can see that, even with our slowest of the optimized functions, we will easily be able to meet the time constraint of an 8kHz sampling rate. (The unoptimized functions, however, will struggle!)

Part 2:

Instructions:

1. Clone repo or download files.
2. In stm32programmer Download "combined.bin" to address 0x8020000
3. Import project to the stm32ide.
4. Flash device (Optional: Comment out or include the OPTIMIZED define)
5. In stm32programmer at address 0x802f000 with size 0xf000 download the data as "test.bin"
6. Run "binary_to_image.py"

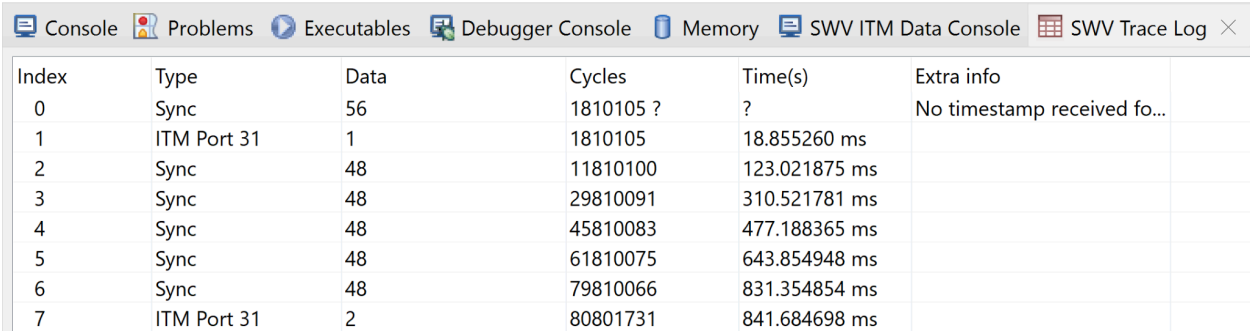
7. Repeat steps 2-6 if you need to rebuild and/or generate a different image (Changes may need to be made to the code if less than or greater than 5 images are put in or if the size of each image is changed from 64x64 pixels)

Implementation:

Following the Kernel (image processing) wiki, we decided to use the crop/avoid overlap method for our edge handling and as a result, there is a 1px black border around the filtered images. To prevent overflow and underflow, we accumulated values in an int32_t variable and clamped the final sum to the valid RGB range (0–255) before storing the result. Filtered pixel values were written to flash memory using HAL_FLASH_Program(), and we verified that stored data using the STM32CubeProgrammer by inspecting the expected memory locations. Testing of the filter was conducted in multiple stages:

- Initially, we manually verified pixel values by printing them and comparing them to expected results.
- Further validation was done using a Python script. input.txt and output.txt were used to cross-check expected vs. actual filtered values.
- Finally, development testing generated test2.bin and test3.bin, which stored filtered image outputs using sharpen and edge detection kernels, respectively.

Baseline Image Filter Statistics



Index	Type	Data	Cycles	Time(s)	Extra info
0	Sync	56	1810105 ?	?	No timestamp received fo...
1	ITM Port 31	1	1810105	18.855260 ms	
2	Sync	48	11810100	123.021875 ms	
3	Sync	48	29810091	310.521781 ms	
4	Sync	48	45810083	477.188365 ms	
5	Sync	48	61810075	643.854948 ms	
6	Sync	48	79810066	831.354854 ms	
7	ITM Port 31	2	80801731	841.684698 ms	

$841.684698ms - 18.855260ms = 0.822829438 = \sim 0.823s$

$80,801,731\ cycles - 1,810,105\ cycles = 78,991,626\ cycles$

*** Below is the implementation of our initial image filter

```
void image_filter(volatile uint8_t *image_addr, volatile uint8_t *output_addr)
{uint8_t buffer[WIDTH * HEIGHT * CHANNELS] = {0x00};
 for (int y = 1; y < HEIGHT - 1; y++)
```

```
 {
   for (int x = 1; x < WIDTH - 1; x++)
   {
     for (int c = 0; c < CHANNELS; c++)
     {
       int sum = 0;
       for (int ky = -1; ky <= 1; ky++)
       {
         for (int kx = -1; kx <= 1; kx++)
         {
           sum += (int)(image_addr[((y + ky) * WIDTH + (x + kx)) * CHANNELS + c] *
kernel[ky + 1][kx + 1]);
         }
       }
       // Clamp the result to the valid range [0, 255]
       if (sum < 0)
         sum = 0;
       if (sum > 255)
         sum = 255;
       buffer[(y * WIDTH + x) * CHANNELS + c] = (uint8_t)sum;
     }
   }
 }
 // After processing the entire image, write the buffer to flash
 HAL_FLASH_Unlock();
 for (int i = 0; i < sizeof(buffer); i++) {
        HAL_FLASH_Program_IT(FLASH_TYPEPROGRAM_BYTE, (uint32_t) (output_addr + i),
buffer[i]);
 }
 HAL_FLASH_Lock();
}
```

*** End of function

Potential Bottlenecks:

The main possible bottlenecks in our function are deep loop nesting, possible inefficient memory access causing cache misses, and frequent byte-wise flash writes.

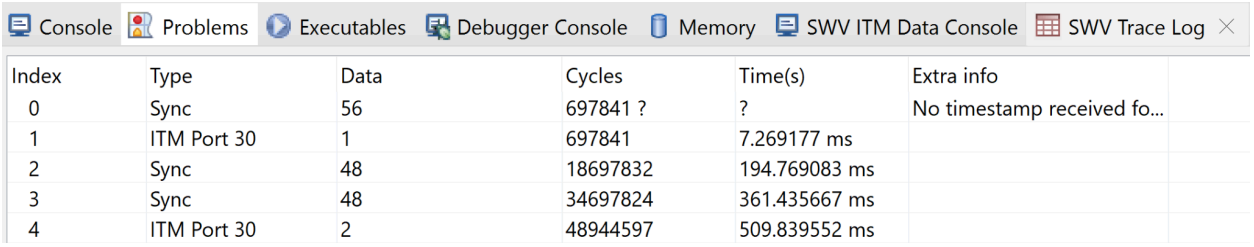
*** Below is the implementation of our optimized image filter

```
void optimized_image_filter(volatile uint8_t *image_addr, volatile uint8_t
*output_addr)
{
      uint8_t buffer[WIDTH * HEIGHT * CHANNELS] = {0x00};
 for (int y = 1; y < HEIGHT - 1; y++)
 {
   for (int x = 1; x < WIDTH - 1; x++)
   {
     for (int c = 0; c < CHANNELS; c++)
     {
       int sum = 0;
       // Unroll the kernel loops
       sum += (int)(image_addr[((y - 1) * WIDTH + (x - 1)) * CHANNELS + c] *
kernel[0][0]);
       sum += (int)(image_addr[((y - 1) * WIDTH + (x)) * CHANNELS + c] *
kernel[0][1]);
       sum += (int)(image_addr[((y - 1) * WIDTH + (x + 1)) * CHANNELS + c] *
kernel[0][2]);
       sum += (int)(image_addr[((y)*WIDTH + (x - 1)) * CHANNELS + c] * kernel[1][0]);
       sum += (int)(image_addr[((y)*WIDTH + (x)) * CHANNELS + c] * kernel[1][1]);
       sum += (int)(image_addr[((y)*WIDTH + (x + 1)) * CHANNELS + c] * kernel[1][2]);
       sum += (int)(image_addr[((y + 1) * WIDTH + (x - 1)) * CHANNELS + c] *
kernel[2][0]);
       sum += (int)(image_addr[((y + 1) * WIDTH + (x)) * CHANNELS + c] *
kernel[2][1]);
       sum += (int)(image_addr[((y + 1) * WIDTH + (x + 1)) * CHANNELS + c] *
kernel[2][2]);
       // Clamp the result to the valid range [0, 255]
       if (sum < 0)
         sum = 0;
       if (sum > 255)
```

```
            sum = 255;
          buffer[(y * WIDTH + x) * CHANNELS + c] = (uint8_t)sum;
        }
      }
    }
  // After processing the entire image, write the buffer to flash
  HAL_FLASH_Unlock();
  for (int i = 0; i < sizeof(buffer); i++) {
          HAL_FLASH_Program_IT(FLASH_TYPEPROGRAM_BYTE, (uint32_t) (output_addr + i),
buffer[i]);
  }
  HAL_FLASH_Lock();
}
```

*** End of function

Optimized Image Filter Statistics



Index	Type	Data	Cycles	Time(s)	Extra info
0	Sync	56	697841 ?	?	No timestamp received fo...
1	ITM Port 30	1	697841	7.269177 ms	
2	Sync	48	18697832	194.769083 ms	
3	Sync	48	34697824	361.435667 ms	
4	ITM Port 30	2	48944597	509.839552 ms	

$509.839552ms - 7.269177ms = 0.502570375s = \sim 0.503s$

$48,944,597\ cycles - 697,841\ cycles = 48,246,756\ cycles$

Difference Between Filter Statistics

$\frac{\sim 0.823s - \sim 0.503s}{\sim 0.823s} \times 100\% = \sim 38.88\%\ time\ reduction$

$\frac{78{,}991{,}626\ cycles - 48{,}246{,}756\ cycles}{78{,}991{,}626\ cycles} \times 100\% = \sim 38.92\%\ cycle\ reduction$

Walkthrough of different function implementations:

When thinking of implementations we came up with examining which values around would utilize its values and add the calculated value to those surrounding values but that ended up being complicated and tedious to think of the logic. So we ended up going through each pixel that will be filtered and applying the kernel one pixel at a time. Since the external borders would not be able to be filtered properly we decided to just ignore those values and replace them with black (0x00). Writing directly to memory was creating artifacts in the image so we saved it to a buffer before transferring it to memory which was saved at address 0x802f000 which is 5 64x64 images apart from where we had saved the original images. The clamping was simple since the values of each pixel were unsigned when they were put into the filter the calculation became straightforward and required clamping for numbers less than 0 or greater than 255. When optimizing the code we had unrolled the loops for the kernel which seemed like the simplest way to unroll the code without making the code tedious or overly large.

Explanation of the bottlenecks in our initial implementation:

The filter calculation could be optimized using a MAC instruction to reduce clock cycles. The multiple nested loops prevent efficient caching and parallel processing, limiting performance gains. Additionally, redundant computations occur when applying the kernel, as operations are performed even when the kernel value is zero, leading to unnecessary processing overhead.

Thoughts and justifications for trying different optimization approaches

Initially, we experimented with various MAC instructions to optimize our code, aiming to enhance computational speed. However, after some initial testing, we found that either our filter was no longer working as expected or there were no consistent noticeable improvements in speed. Another option of optimization we considered is loop unrolling. It reduces the overhead associated with loop control instructions, such as incrementing certain counters or checking loop end conditions. Instead of executing the loop one iteration at a time, multiple iterations are performed within a single loop iteration. Some benefits is that, like mentioned earlier; it reduces loop overhead, but not only that, it can better take advantage of parallel processing. Some drawbacks to this method is that it increases the code size and there can be diminishing returns if excessively used. After evaluating these strategies, we decided to implement loop unrolling as we thought that its pros outweigh the cons for what we wanted to do with it. Optimizing our filter this way improved the execution speed by ~38.88% and reduced the amount of cycles needed by ~38.92% which is quite considerable.

*Measurements taken as part of our analysis of the code are done above.

Appendix – Function Disassembly

```
567       static int16_t ProcessSampleCircular(int16_t newsample, int16_t* history) {
    ProcessSampleCircular:
08001264:   push    {r7}
08001266:   sub     sp, #28
08001268:   add     r7, sp, #0
0800126a:   mov     r3, r0
0800126c:   str     r1, [r7, #0]
0800126e:   strh    r3, [r7, #6]
570         history[head] = newsample;
08001270:   ldr     r3, [pc, #220]  @ (0x8001350 <ProcessSampleCircular+236>)
08001272:   ldr     r3, [r3, #0]
08001274:   lsls    r3, r3, #1
08001276:   ldr     r2, [r7, #0]
08001278:   add     r3, r2
0800127a:   ldrh    r2, [r7, #6]
0800127c:   strh    r2, [r3, #0]
573         int tap = 0; // indexing filter_coeffs
0800127e:   movs    r3, #0
08001280:   str     r3, [r7, #20]
574         int current = head; //  indexing history array
08001282:   ldr     r3, [pc, #204]  @ (0x8001350 <ProcessSampleCircular+236>)
08001284:   ldr     r3, [r3, #0]
08001286:   str     r3, [r7, #16]
575         int32_t accumulator = 0;
08001288:   movs    r3, #0
0800128a:   str     r3, [r7, #12]
576         for (tap = 0, current = head; tap < NUMBER_OF_TAPS; tap++, current--) {
0800128c:   movs    r3, #0
0800128e:   str     r3, [r7, #20]
08001290:   ldr     r3, [pc, #188]  @ (0x8001350 <ProcessSampleCircular+236>)
08001292:   ldr     r3, [r3, #0]
08001294:   str     r3, [r7, #16]
08001296:   b.n     0x80012d6 <ProcessSampleCircular+114>
577             current = (current + NUMBER_OF_TAPS) % NUMBER_OF_TAPS;
08001298:   ldr     r3, [r7, #16]
0800129a:   add.w   r3, r3, #256    @ 0x100
0800129e:   negs    r2, r3
080012a0:   uxtb    r3, r3
080012a2:   uxtb    r2, r2
080012a4:   it      pl
080012a6:   negpl   r3, r2
080012a8:   str     r3, [r7, #16]
578             accumulator += (int32_t)filter_coeffs[tap] * (int32_t)history[current];
080012aa:   ldr     r2, [pc, #168]  @ (0x8001354 <ProcessSampleCircular+240>)
080012ac:   ldr     r3, [r7, #20]
080012ae:   ldrsh.w r3, [r2, r3, lsl #1]
080012b2:   mov     r1, r3
080012b4:   ldr     r3, [r7, #16]
080012b6:   lsls    r3, r3, #1
080012b8:   ldr     r2, [r7, #0]
080012ba:   add     r3, r2
080012bc:   ldrsh.w r3, [r3]
080012c0:   mul.w   r3, r1, r3
080012c4:   ldr     r2, [r7, #12]
080012c6:   add     r3, r2
080012c8:   str     r3, [r7, #12]
576         for (tap = 0, current = head; tap < NUMBER_OF_TAPS; tap++, current--) {
080012ca:   ldr     r3, [r7, #20]
080012cc:   adds    r3, #1
080012ce:   str     r3, [r7, #20]
080012d0:   ldr     r3, [r7, #16]
080012d2:   subs    r3, #1
080012d4:   str     r3, [r7, #16]
080012d6:   ldr     r3, [r7, #20]
080012d8:   cmp     r3, #255        @ 0xff
080012da:   ble.n   0x8001298 <ProcessSampleCircular+52>
582         head = (head + 1) % NUMBER_OF_TAPS;
080012dc:   ldr     r3, [pc, #112]  @ (0x8001350 <ProcessSampleCircular+236>)
080012de:   ldr     r3, [r3, #0]
080012e0:   adds    r3, #1
080012e2:   negs    r2, r3
080012e4:   uxtb    r3, r3
080012e6:   uxtb    r2, r2
080012e8:   it      pl
```

```
080012ea:   negpl   r3, r2
080012ec:   ldr     r2, [pc, #96]   @ (0x8001350 <ProcessSampleCircular+236>)
080012ee:   str     r3, [r2, #0]
586         if (accumulator > 0x3FFFFFFF) {
080012f0:   ldr     r3, [r7, #12]
080012f2:   cmp.w   r3, #1073741824 @ 0x40000000
080012f6:   blt.n   0x800130a <ProcessSampleCircular+166>
587             accumulator = 0x3FFFFFFF;
080012f8:   mvn.w   r3, #3221225472 @ 0xc0000000
080012fc:   str     r3, [r7, #12]
588             overflow_count++;
080012fe:   ldr     r3, [pc, #88]   @ (0x8001358 <ProcessSampleCircular+244>)
08001300:   ldr     r3, [r3, #0]
08001302:   adds    r3, #1
08001304:   ldr     r2, [pc, #80]   @ (0x8001358 <ProcessSampleCircular+244>)
08001306:   str     r3, [r2, #0]
08001308:   b.n     0x8001322 <ProcessSampleCircular+190>
589         } else if (accumulator < -0x40000000) {
0800130a:   ldr     r3, [r7, #12]
0800130c:   cmp.w   r3, #3221225472 @ 0xc0000000
08001310:   bge.n   0x8001322 <ProcessSampleCircular+190>
590             accumulator = -0x40000000;
08001312:   mov.w   r3, #3221225472 @ 0xc0000000
08001316:   str     r3, [r7, #12]
591             underflow_count++;
08001318:   ldr     r3, [pc, #64]   @ (0x800135c <ProcessSampleCircular+248>)
0800131a:   ldr     r3, [r3, #0]
0800131c:   adds    r3, #1
0800131e:   ldr     r2, [pc, #60]   @ (0x800135c <ProcessSampleCircular+248>)
08001320:   str     r3, [r2, #0]
595         if (accumulator < 1000 && accumulator > -1000) {
08001322:   ldr     r3, [r7, #12]
08001324:   cmp.w   r3, #1000       @ 0x3e8
08001328:   bge.n   0x8001338 <ProcessSampleCircular+212>
0800132a:   ldr     r3, [r7, #12]
0800132c:   cmn.w   r3, #1000       @ 0x3e8
08001330:   ble.n   0x8001338 <ProcessSampleCircular+212>
596             temp = (int16_t)accumulator;
08001332:   ldr     r3, [r7, #12]
08001334:   strh    r3, [r7, #10]
08001336:   b.n     0x800133e <ProcessSampleCircular+218>
598             temp = (int16_t)(accumulator >> 15);
08001338:   ldr     r3, [r7, #12]
0800133a:   asrs    r3, r3, #15
0800133c:   strh    r3, [r7, #10]
600         return temp;
0800133e:   ldrsh.w r3, [r7, #10]
601     }
08001342:   mov     r0, r3
08001344:   adds    r7, #28
08001346:   mov     sp, r7
08001348:   ldr.w   r7, [sp], #4
0800134c:   bx      lr
```

```
605     static int16_t ProcessSampleCircular2(int16_t newsample, int16_t* history) {
        ProcessSampleCircular2:
08001264:   push    {r7}
08001266:   sub     sp, #36 @ 0x24
08001268:   add     r7, sp, #0
0800126a:   mov     r3, r0
0800126c:   str     r1, [r7, #0]
0800126e:   strh    r3, [r7, #6]
608           history[head] = newsample;
08001270:   ldr     r3, [pc, #276]  @ (0x8001388 <ProcessSampleCircular2+292>)
08001272:   ldr     r3, [r3, #0]
08001274:   lsls    r3, r3, #1
08001276:   ldr     r2, [r7, #0]
08001278:   add     r3, r2
0800127a:   ldrh    r2, [r7, #6]
0800127c:   strh    r2, [r3, #0]
611           int tap = 0; // indexing filter_coeffs
0800127e:   movs    r3, #0
08001280:   str     r3, [r7, #28]
612           int current = head; // indexing history array
08001282:   ldr     r3, [pc, #260]  @ (0x8001388 <ProcessSampleCircular2+292>)
08001284:   ldr     r3, [r3, #0]
08001286:   str     r3, [r7, #16]
613           int32_t accumulator = 0;
```

```
08001288:   movs    r3, #0
0800128a:   str     r3, [r7, #24]
616           for (tap = 0; tap < (NUMBER_OF_TAPS / 2) * 2; tap += 2) {
0800128c:   movs    r3, #0
0800128e:   str     r3, [r7, #28]
08001290:   b.n     0x8001310 <ProcessSampleCircular2+172>
618               current = (head - tap) % NUMBER_OF_TAPS;
08001292:   ldr     r3, [pc, #244]  @ (0x8001388 <ProcessSampleCircular2+292>)
08001294:   ldr     r2, [r3, #0]
08001296:   ldr     r3, [r7, #28]
08001298:   subs    r3, r2, r3
0800129a:   negs    r2, r3
0800129c:   uxtb    r3, r3
0800129e:   uxtb    r2, r2
080012a0:   it      pl
080012a2:   negpl   r3, r2
080012a4:   str     r3, [r7, #16]
619               int history_idx_1 = (current + NUMBER_OF_TAPS - 1) % NUMBER_OF_TAPS;
080012a6:   ldr     r3, [r7, #16]
080012a8:   adds    r3, #255        @ 0xff
080012aa:   negs    r2, r3
080012ac:   uxtb    r3, r3
080012ae:   uxtb    r2, r2
080012b0:   it      pl
080012b2:   negpl   r3, r2
080012b4:   str     r3, [r7, #12]
620               int history_idx_2 = (current + NUMBER_OF_TAPS) % NUMBER_OF_TAPS;
080012b6:   ldr     r3, [r7, #16]
080012b8:   add.w   r3, r3, #256    @ 0x100
080012bc:   negs    r2, r3
080012be:   uxtb    r3, r3
080012c0:   uxtb    r2, r2
080012c2:   it      pl
080012c4:   negpl   r3, r2
080012c6:   str     r3, [r7, #8]
625                               : [op1] "r" ((int32_t)filter_coeffs[tap]),
080012c8:   ldr     r2, [pc, #192]  @ (0x800138c <ProcessSampleCircular2+296>)
080012ca:   ldr     r3, [r7, #28]
080012cc:   ldrsh.w r3, [r2, r3, lsl #1]
080012d0:   mov     r1, r3
626                                 [op2] "r" ((int32_t)history[history_idx_2]),
080012d2:   ldr     r3, [r7, #8]
080012d4:   lsls    r3, r3, #1
080012d6:   ldr     r2, [r7, #0]
080012d8:   add     r3, r2
080012da:   ldrsh.w r3, [r3]
080012de:   mov     r2, r3
623               __asm volatile ("SMLABB %[result1], %[op1], %[op2], %[acc1]"
080012e0:   ldr     r3, [r7, #24]
080012e2:   smlabb  r3, r1, r2, r3
080012e6:   str     r3, [r7, #24]
633                               : [op1] "r" ((int32_t)filter_coeffs[tap + 1]),
080012e8:   ldr     r3, [r7, #28]
080012ea:   adds    r3, #1
080012ec:   ldr     r2, [pc, #156]  @ (0x800138c <ProcessSampleCircular2+296>)
080012ee:   ldrsh.w r3, [r2, r3, lsl #1]
080012f2:   mov     r1, r3
634                                 [op2] "r" ((int32_t)history[history_idx_1]),
080012f4:   ldr     r3, [r7, #12]
080012f6:   lsls    r3, r3, #1
080012f8:   ldr     r2, [r7, #0]
080012fa:   add     r3, r2
080012fc:   ldrsh.w r3, [r3]
08001300:   mov     r2, r3
631               __asm volatile ("SMLABB %[result2], %[op1], %[op2], %[acc2]"
08001302:   ldr     r3, [r7, #24]
08001304:   smlabb  r3, r1, r2, r3
08001308:   str     r3, [r7, #24]
0800130a:   ldr     r3, [r7, #28]
0800130c:   adds    r3, #2
0800130e:   str     r3, [r7, #28]
08001310:   ldr     r3, [r7, #28]
08001312:   cmp     r3, #255        @ 0xff
08001314:   ble.n   0x8001292 <ProcessSampleCircular2+46>
651           head = (head + 1) % NUMBER_OF_TAPS;
08001316:   ldr     r3, [pc, #112]  @ (0x8001388 <ProcessSampleCircular2+292>)
08001318:   ldr     r3, [r3, #0]
0800131a:   adds    r3, #1
0800131c:   negs    r2, r3
```

```
0800131e:   uxtb    r3, r3
08001320:   uxtb    r2, r2
08001322:   it      pl
08001324:   negpl   r3, r2
08001326:   ldr     r2, [pc, #96]   @ (0x8001388 <ProcessSampleCircular2+292>)
08001328:   str     r3, [r2, #0]
654           if (accumulator > 0x3FFFFFFF) {
0800132a:   ldr     r3, [r7, #24]
0800132c:   cmp.w   r3, #1073741824 @ 0x40000000
08001330:   blt.n   0x8001344 <ProcessSampleCircular2+224>
655               accumulator = 0x3FFFFFFF;
08001332:   mvn.w   r3, #3221225472 @ 0xc0000000
08001336:   str     r3, [r7, #24]
656               overflow_count++;
08001338:   ldr     r3, [pc, #84]   @ (0x8001390 <ProcessSampleCircular2+300>)
0800133a:   ldr     r3, [r3, #0]
0800133c:   adds    r3, #1
0800133e:   ldr     r2, [pc, #80]   @ (0x8001390 <ProcessSampleCircular2+300>)
08001340:   str     r3, [r2, #0]
08001342:   b.n     0x800135c <ProcessSampleCircular2+248>
657           } else if (accumulator < -0x40000000) {
08001344:   ldr     r3, [r7, #24]
08001346:   cmp.w   r3, #3221225472 @ 0xc0000000
0800134a:   bge.n   0x800135c <ProcessSampleCircular2+248>
658               accumulator = -0x40000000;
0800134c:   mov.w   r3, #3221225472 @ 0xc0000000
08001350:   str     r3, [r7, #24]
659               underflow_count++;
08001352:   ldr     r3, [pc, #64]   @ (0x8001394 <ProcessSampleCircular2+304>)
08001354:   ldr     r3, [r3, #0]
08001356:   adds    r3, #1
08001358:   ldr     r2, [pc, #56]   @ (0x8001394 <ProcessSampleCircular2+304>)
0800135a:   str     r3, [r2, #0]
664           if (accumulator < 1000 && accumulator > -1000) {
0800135c:   ldr     r3, [r7, #24]
0800135e:   cmp.w   r3, #1000       @ 0x3e8
08001362:   bge.n   0x8001372 <ProcessSampleCircular2+270>
08001364:   ldr     r3, [r7, #24]
08001366:   cmn.w   r3, #1000       @ 0x3e8
0800136a:   ble.n   0x8001372 <ProcessSampleCircular2+270>
665               temp = (int16_t)accumulator;
0800136c:   ldr     r3, [r7, #24]
0800136e:   strh    r3, [r7, #22]
08001370:   b.n     0x8001378 <ProcessSampleCircular2+276>
667               temp = (int16_t)(accumulator >> 15);
08001372:   ldr     r3, [r7, #24]
08001374:   asrs    r3, r3, #15
08001376:   strh    r3, [r7, #22]
670           return temp;
08001378:   ldrsh.w r3, [r7, #22]
671       }
0800137c:   mov     r0, r3
0800137e:   adds    r7, #36 @ 0x24
08001380:   mov     sp, r7
08001382:   ldr.w   r7, [sp], #4
08001386:   bx      lr
```

```
673       static int  ProcessBlock(int16_t newsample, int16_t* history) {
          ProcessBlock:
08001274:   push    {r7}
08001276:   sub     sp, #52 @ 0x34
08001278:   add     r7, sp, #0
0800127a:   mov     r3, r0
0800127c:   str     r1, [r7, #0]
0800127e:   strh    r3, [r7, #6]
675         history[head] = newsample;
08001280:   ldr     r3, [pc, #488]  @ (0x800146c <ProcessBlock+504>)
08001282:   ldr     r3, [r3, #0]
08001284:   lsls    r3, r3, #1
08001286:   ldr     r2, [r7, #0]
08001288:   add     r3, r2
0800128a:   ldrh    r2, [r7, #6]
0800128c:   strh    r2, [r3, #0]
676         if (samples_since_last_frame++ < FRAME_SIZE - 1 ){
```

```
0800128e:   ldr     r3, [pc, #480]  @ (0x8001470 <ProcessBlock+508>)
08001290:   ldr     r3, [r3, #0]
08001292:   adds    r2, r3, #1
08001294:   ldr     r1, [pc, #472]  @ (0x8001470 <ProcessBlock+508>)
08001296:   str     r2, [r1, #0]
08001298:   cmp     r3, #1
0800129a:   bgt.n   0x80012c2 <ProcessBlock+78>
677             head = (head + 1) % HISTORY_SIZE;
0800129c:   ldr     r3, [pc, #460]  @ (0x800146c <ProcessBlock+504>)
0800129e:   ldr     r3, [r3, #0]
080012a0:   adds    r1, r3, #1
080012a2:   ldr     r3, [pc, #464]  @ (0x8001474 <ProcessBlock+512>)
080012a4:   smull   r2, r3, r3, r1
080012a8:   asrs    r2, r3, #7
080012aa:   asrs    r3, r1, #31
080012ac:   subs    r2, r2, r3
080012ae:   mov     r3, r2
080012b0:   lsls    r3, r3, #7
080012b2:   add     r3, r2
080012b4:   lsls    r3, r3, #1
080012b6:   add     r3, r2
080012b8:   subs    r2, r1, r3
080012ba:   ldr     r3, [pc, #432]  @ (0x800146c <ProcessBlock+504>)
080012bc:   str     r2, [r3, #0]
679             return false;
080012be:   movs    r3, #0
080012c0:   b.n     0x8001460 <ProcessBlock+492>
682         samples_since_last_frame = 0;
080012c2:   ldr     r3, [pc, #428]  @ (0x8001470 <ProcessBlock+508>)
080012c4:   movs    r2, #0
080012c6:   str     r2, [r3, #0]
687         int32_t accumulators [FRAME_SIZE] = {0}; // accumulator[2] corresponds to the newest sample
080012c8:   add.w   r3, r7, #8
080012cc:   movs    r2, #0
080012ce:   str     r2, [r3, #0]
080012d0:   str     r2, [r3, #4]
080012d2:   str     r2, [r3, #8]
688         for (tap = 0, current = head; tap < NUMBER_OF_TAPS; tap++, current--) {
080012d4:   movs    r3, #0
080012d6:   str     r3, [r7, #44]   @ 0x2c
080012d8:   ldr     r3, [pc, #400]  @ (0x800146c <ProcessBlock+504>)
080012da:   ldr     r3, [r3, #0]
080012dc:   str     r3, [r7, #40]   @ 0x28
080012de:   b.n     0x8001376 <ProcessBlock+258>
689                 int32_t coeff = (int32_t)filter_coeffs[tap]; // loading the coefficient once
080012e0:   ldr     r2, [pc, #404]  @ (0x8001478 <ProcessBlock+516>)
080012e2:   ldr     r3, [r7, #44]   @ 0x2c
080012e4:   ldrsh.w r3, [r2, r3, lsl #1]
080012e8:   str     r3, [r7, #28]
691                 int base_idx = (current - MAX_FRAME_IDX + HISTORY_SIZE) % HISTORY_SIZE;
080012ea:   ldr     r3, [r7, #40]   @ 0x28
080012ec:   addw    r2, r3, #257    @ 0x101
080012f0:   ldr     r3, [pc, #384]  @ (0x8001474 <ProcessBlock+512>)
080012f2:   smull   r1, r3, r3, r2
080012f6:   asrs    r1, r3, #7
080012f8:   asrs    r3, r2, #31
080012fa:   subs    r1, r1, r3
080012fc:   mov     r3, r1
080012fe:   lsls    r3, r3, #7
08001300:   add     r3, r1
08001302:   lsls    r3, r3, #1
08001304:   add     r3, r1
08001306:   subs    r3, r2, r3
08001308:   str     r3, [r7, #24]
693                 for (int i = 0; i < FRAME_SIZE; i++){
0800130a:   movs    r3, #0
0800130c:   str     r3, [r7, #36]   @ 0x24
0800130e:   b.n     0x8001364 <ProcessBlock+240>
694                     int history_idx = (base_idx + i) % HISTORY_SIZE;
08001310:   ldr     r2, [r7, #24]
08001312:   ldr     r3, [r7, #36]   @ 0x24
08001314:   add     r2, r3
08001316:   ldr     r3, [pc, #348]  @ (0x8001474 <ProcessBlock+512>)
08001318:   smull   r1, r3, r3, r2
0800131c:   asrs    r1, r3, #7
0800131e:   asrs    r3, r2, #31
08001320:   subs    r1, r1, r3
08001322:   mov     r3, r1
08001324:   lsls    r3, r3, #7
```

```
08001326:   add     r3, r1
08001328:   lsls    r3, r3, #1
0800132a:   add     r3, r1
0800132c:   subs    r3, r2, r3
0800132e:   str     r3, [r7, #20]
695                     accumulators[i] += coeff * (int32_t)history[history_idx];
08001330:   ldr     r3, [r7, #36]   @ 0x24
08001332:   lsls    r3, r3, #2
08001334:   adds    r3, #48 @ 0x30
08001336:   add     r3, r7
08001338:   ldr.w   r2, [r3, #-40]
0800133c:   ldr     r3, [r7, #20]
0800133e:   lsls    r3, r3, #1
08001340:   ldr     r1, [r7, #0]
08001342:   add     r3, r1
08001344:   ldrsh.w r3, [r3]
08001348:   mov     r1, r3
0800134a:   ldr     r3, [r7, #28]
0800134c:   mul.w   r3, r1, r3
08001350:   add     r2, r3
08001352:   ldr     r3, [r7, #36]   @ 0x24
08001354:   lsls    r3, r3, #2
08001356:   adds    r3, #48 @ 0x30
08001358:   add     r3, r7
0800135a:   str.w   r2, [r3, #-40]
693                 for (int i = 0; i < FRAME_SIZE; i++){
0800135e:   ldr     r3, [r7, #36]   @ 0x24
08001360:   adds    r3, #1
08001362:   str     r3, [r7, #36]   @ 0x24
08001364:   ldr     r3, [r7, #36]   @ 0x24
08001366:   cmp     r3, #2
08001368:   ble.n   0x8001310 <ProcessBlock+156>
688         for (tap = 0, current = head; tap < NUMBER_OF_TAPS; tap++, current--) {
0800136a:   ldr     r3, [r7, #44]   @ 0x2c
0800136c:   adds    r3, #1
0800136e:   str     r3, [r7, #44]   @ 0x2c
08001370:   ldr     r3, [r7, #40]   @ 0x28
08001372:   subs    r3, #1
08001374:   str     r3, [r7, #40]   @ 0x28
08001376:   ldr     r3, [r7, #44]   @ 0x2c
08001378:   cmp     r3, #255        @ 0xff
0800137a:   ble.n   0x80012e0 <ProcessBlock+108>
701         head = (head + 1) % HISTORY_SIZE; // moving the head
0800137c:   ldr     r3, [pc, #236]  @ (0x800146c <ProcessBlock+504>)
0800137e:   ldr     r3, [r3, #0]
08001380:   adds    r1, r3, #1
08001382:   ldr     r3, [pc, #240]  @ (0x8001474 <ProcessBlock+512>)
08001384:   smull   r2, r3, r3, r1
08001388:   asrs    r2, r3, #7
0800138a:   asrs    r3, r1, #31
0800138c:   subs    r2, r2, r3
0800138e:   mov     r3, r2
08001390:   lsls    r3, r3, #7
08001392:   add     r3, r2
08001394:   lsls    r3, r3, #1
08001396:   add     r3, r2
08001398:   subs    r2, r1, r3
0800139a:   ldr     r3, [pc, #208]  @ (0x800146c <ProcessBlock+504>)
0800139c:   str     r2, [r3, #0]
704         for (int i = 0; i < FRAME_SIZE; i++){
0800139e:   movs    r3, #0
080013a0:   str     r3, [r7, #32]
080013a2:   b.n     0x8001458 <ProcessBlock+484>
705             if (accumulators[i] > 0x3FFFFFFF) {
080013a4:   ldr     r3, [r7, #32]
080013a6:   lsls    r3, r3, #2
080013a8:   adds    r3, #48 @ 0x30
080013aa:   add     r3, r7
080013ac:   ldr.w   r3, [r3, #-40]
080013b0:   cmp.w   r3, #1073741824 @ 0x40000000
080013b4:   blt.n   0x80013d2 <ProcessBlock+350>
706                     accumulators[i]= 0x3FFFFFFF;
080013b6:   ldr     r3, [r7, #32]
080013b8:   lsls    r3, r3, #2
080013ba:   adds    r3, #48 @ 0x30
080013bc:   add     r3, r7
080013be:   mvn.w   r2, #3221225472 @ 0xc0000000
080013c2:   str.w   r2, [r3, #-40]
707                     overflow_count++;
```

```
080013c6:   ldr     r3, [pc, #180]  @ (0x800147c <ProcessBlock+520>)
080013c8:   ldr     r3, [r3, #0]
080013ca:   adds    r3, #1
080013cc:   ldr     r2, [pc, #172]  @ (0x800147c <ProcessBlock+520>)
080013ce:   str     r3, [r2, #0]
080013d0:   b.n     0x80013fe <ProcessBlock+394>
708                 } else if (accumulators[i] < -0x40000000) {
080013d2:   ldr     r3, [r7, #32]
080013d4:   lsls    r3, r3, #2
080013d6:   adds    r3, #48 @ 0x30
080013d8:   add     r3, r7
080013da:   ldr.w   r3, [r3, #-40]
080013de:   cmp.w   r3, #3221225472 @ 0xc0000000
080013e2:   bge.n   0x80013fe <ProcessBlock+394>
709                     accumulators[i] = -0x40000000;
080013e4:   ldr     r3, [r7, #32]
080013e6:   lsls    r3, r3, #2
080013e8:   adds    r3, #48 @ 0x30
080013ea:   add     r3, r7
080013ec:   mov.w   r2, #3221225472 @ 0xc0000000
080013f0:   str.w   r2, [r3, #-40]
710                     underflow_count++;
080013f4:   ldr     r3, [pc, #136]  @ (0x8001480 <ProcessBlock+524>)
080013f6:   ldr     r3, [r3, #0]
080013f8:   adds    r3, #1
080013fa:   ldr     r2, [pc, #132]  @ (0x8001480 <ProcessBlock+524>)
080013fc:   str     r3, [r2, #0]
713             if (accumulators[i] < 1000 && accumulators[i] > -1000) {
080013fe:   ldr     r3, [r7, #32]
08001400:   lsls    r3, r3, #2
08001402:   adds    r3, #48 @ 0x30
08001404:   add     r3, r7
08001406:   ldr.w   r3, [r3, #-40]
0800140a:   cmp.w   r3, #1000       @ 0x3e8
0800140e:   bge.n   0x800143a <ProcessBlock+454>
08001410:   ldr     r3, [r7, #32]
08001412:   lsls    r3, r3, #2
08001414:   adds    r3, #48 @ 0x30
08001416:   add     r3, r7
08001418:   ldr.w   r3, [r3, #-40]
0800141c:   cmn.w   r3, #1000       @ 0x3e8
08001420:   ble.n   0x800143a <ProcessBlock+454>
714                 accumulators_16[i] = (int16_t)accumulators[i];
08001422:   ldr     r3, [r7, #32]
08001424:   lsls    r3, r3, #2
08001426:   adds    r3, #48 @ 0x30
08001428:   add     r3, r7
0800142a:   ldr.w   r3, [r3, #-40]
0800142e:   sxth    r1, r3
08001430:   ldr     r2, [pc, #80]   @ (0x8001484 <ProcessBlock+528>)
08001432:   ldr     r3, [r7, #32]
08001434:   strh.w  r1, [r2, r3, lsl #1]
08001438:   b.n     0x8001452 <ProcessBlock+478>
716                 accumulators_16[i]= (int16_t)(accumulators[i] >> 15);
0800143a:   ldr     r3, [r7, #32]
0800143c:   lsls    r3, r3, #2
0800143e:   adds    r3, #48 @ 0x30
08001440:   add     r3, r7
08001442:   ldr.w   r3, [r3, #-40]
08001446:   asrs    r3, r3, #15
08001448:   sxth    r1, r3
0800144a:   ldr     r2, [pc, #56]   @ (0x8001484 <ProcessBlock+528>)
0800144c:   ldr     r3, [r7, #32]
0800144e:   strh.w  r1, [r2, r3, lsl #1]
704         for (int i = 0; i < FRAME_SIZE; i++){
08001452:   ldr     r3, [r7, #32]
08001454:   adds    r3, #1
08001456:   str     r3, [r7, #32]
08001458:   ldr     r3, [r7, #32]
0800145a:   cmp     r3, #2
0800145c:   ble.n   0x80013a4 <ProcessBlock+304>
723         return true;
0800145e:   movs    r3, #1
726     }
08001460:   mov     r0, r3
08001462:   adds    r7, #52 @ 0x34
08001464:   mov     sp, r7
08001466:   ldr.w   r7, [sp], #4
0800146a:   bx      lr
```

```
730       static int ProcessBlock2(int16_t newsample, int16_t* history) {
          ProcessBlock2:
08001274:   push    {r7}
08001276:   sub     sp, #60 @ 0x3c
08001278:   add     r7, sp, #0
0800127a:   mov     r3, r0
0800127c:   str     r1, [r7, #0]
0800127e:   strh    r3, [r7, #6]
732           history[head] = newsample;
08001280:   ldr     r3, [pc, #612]  @ (0x80014e8 <ProcessBlock2+628>)
08001282:   ldr     r3, [r3, #0]
08001284:   lsls    r3, r3, #1
08001286:   ldr     r2, [r7, #0]
08001288:   add     r3, r2
0800128a:   ldrh    r2, [r7, #6]
0800128c:   strh    r2, [r3, #0]
734           if (samples_since_last_frame++ < FRAME_SIZE - 1) {
0800128e:   ldr     r3, [pc, #604]  @ (0x80014ec <ProcessBlock2+632>)
08001290:   ldr     r3, [r3, #0]
08001292:   adds    r2, r3, #1
08001294:   ldr     r1, [pc, #596]  @ (0x80014ec <ProcessBlock2+632>)
08001296:   str     r2, [r1, #0]
08001298:   cmp     r3, #1
0800129a:   bgt.n   0x80012c2 <ProcessBlock2+78>
735               head = (head + 1) % HISTORY_SIZE;
0800129c:   ldr     r3, [pc, #584]  @ (0x80014e8 <ProcessBlock2+628>)
0800129e:   ldr     r3, [r3, #0]
080012a0:   adds    r1, r3, #1
080012a2:   ldr     r3, [pc, #588]  @ (0x80014f0 <ProcessBlock2+636>)
080012a4:   smull   r2, r3, r3, r1
080012a8:   asrs    r2, r3, #7
080012aa:   asrs    r3, r1, #31
080012ac:   subs    r2, r2, r3
080012ae:   mov     r3, r2
080012b0:   lsls    r3, r3, #7
080012b2:   add     r3, r2
080012b4:   lsls    r3, r3, #1
080012b6:   add     r3, r2
080012b8:   subs    r2, r1, r3
080012ba:   ldr     r3, [pc, #556]  @ (0x80014e8 <ProcessBlock2+628>)
080012bc:   str     r2, [r3, #0]
736               return false;
080012be:   movs    r3, #0
080012c0:   b.n     0x80014da <ProcessBlock2+614>
739           samples_since_last_frame = 0;
080012c2:   ldr     r3, [pc, #552]  @ (0x80014ec <ProcessBlock2+632>)
080012c4:   movs    r2, #0
080012c6:   str     r2, [r3, #0]
742           int current = head;
080012c8:   ldr     r3, [pc, #540]  @ (0x80014e8 <ProcessBlock2+628>)
080012ca:   ldr     r3, [r3, #0]
080012cc:   str     r3, [r7, #52]   @ 0x34
743           int32_t accumulators[FRAME_SIZE] = {0};  // accumulator[2] corresponds to the newest sample
080012ce:   add.w   r3, r7, #8
080012d2:   movs    r2, #0
080012d4:   str     r2, [r3, #0]
080012d6:   str     r2, [r3, #4]
080012d8:   str     r2, [r3, #8]
747           for (tap = 0; tap < (NUMBER_OF_TAPS / 2) * 2; tap += 2, current-=2) {
080012da:   movs    r3, #0
080012dc:   str     r3, [r7, #48]   @ 0x30
080012de:   b.n     0x80013ee <ProcessBlock2+378>
748               int32_t coeff1 = (int32_t)filter_coeffs[tap];     // First tap coefficient
080012e0:   ldr     r2, [pc, #528]  @ (0x80014f4 <ProcessBlock2+640>)
080012e2:   ldr     r3, [r7, #48]   @ 0x30
080012e4:   ldrsh.w r3, [r2, r3, lsl #1]
080012e8:   str     r3, [r7, #36]   @ 0x24
749               int32_t coeff2 = (int32_t)filter_coeffs[tap + 1]; // Second tap coefficient
080012ea:   ldr     r3, [r7, #48]   @ 0x30
080012ec:   adds    r3, #1
080012ee:   ldr     r2, [pc, #516]  @ (0x80014f4 <ProcessBlock2+640>)
080012f0:   ldrsh.w r3, [r2, r3, lsl #1]
080012f4:   str     r3, [r7, #32]
750               current = (current + HISTORY_SIZE) % HISTORY_SIZE;
080012f6:   ldr     r3, [r7, #52]   @ 0x34
080012f8:   addw    r2, r3, #259    @ 0x103
080012fc:   ldr     r3, [pc, #496]  @ (0x80014f0 <ProcessBlock2+636>)
```

```
080012fe:   smull   r1, r3, r3, r2
08001302:   asrs    r1, r3, #7
08001304:   asrs    r3, r2, #31
08001306:   subs    r1, r1, r3
08001308:   mov     r3, r1
0800130a:   lsls    r3, r3, #7
0800130c:   add     r3, r1
0800130e:   lsls    r3, r3, #1
08001310:   add     r3, r1
08001312:   subs    r3, r2, r3
08001314:   str     r3, [r7, #52]   @ 0x34
752               int base_idx = (current - FRAME_SIZE + HISTORY_SIZE) % HISTORY_SIZE;
08001316:   ldr     r3, [r7, #52]   @ 0x34
08001318:   add.w   r2, r3, #256    @ 0x100
0800131c:   ldr     r3, [pc, #464]  @ (0x80014f0 <ProcessBlock2+636>)
0800131e:   smull   r1, r3, r3, r2
08001322:   asrs    r1, r3, #7
08001324:   asrs    r3, r2, #31
08001326:   subs    r1, r1, r3
08001328:   mov     r3, r1
0800132a:   lsls    r3, r3, #7
0800132c:   add     r3, r1
0800132e:   lsls    r3, r3, #1
08001330:   add     r3, r1
08001332:   subs    r3, r2, r3
08001334:   str     r3, [r7, #28]
754               for (int i = 0; i < FRAME_SIZE; i++) {
08001336:   movs    r3, #0
08001338:   str     r3, [r7, #44]   @ 0x2c
0800133a:   b.n     0x80013dc <ProcessBlock2+360>
755                 int history_idx_1 = (base_idx + i) % HISTORY_SIZE;
0800133c:   ldr     r2, [r7, #28]
0800133e:   ldr     r3, [r7, #44]   @ 0x2c
08001340:   add     r2, r3
08001342:   ldr     r3, [pc, #428]  @ (0x80014f0 <ProcessBlock2+636>)
08001344:   smull   r1, r3, r3, r2
08001348:   asrs    r1, r3, #7
0800134a:   asrs    r3, r2, #31
0800134c:   subs    r1, r1, r3
0800134e:   mov     r3, r1
08001350:   lsls    r3, r3, #7
08001352:   add     r3, r1
08001354:   lsls    r3, r3, #1
08001356:   add     r3, r1
08001358:   subs    r3, r2, r3
0800135a:   str     r3, [r7, #24]
756                 int history_idx_2 = (base_idx + i + 1) % HISTORY_SIZE;
0800135c:   ldr     r2, [r7, #28]
0800135e:   ldr     r3, [r7, #44]   @ 0x2c
08001360:   add     r3, r2
08001362:   adds    r2, r3, #1
08001364:   ldr     r3, [pc, #392]  @ (0x80014f0 <ProcessBlock2+636>)
08001366:   smull   r1, r3, r3, r2
0800136a:   asrs    r1, r3, #7
0800136c:   asrs    r3, r2, #31
0800136e:   subs    r1, r1, r3
08001370:   mov     r3, r1
08001372:   lsls    r3, r3, #7
08001374:   add     r3, r1
08001376:   lsls    r3, r3, #1
08001378:   add     r3, r1
0800137a:   subs    r3, r2, r3
0800137c:   str     r3, [r7, #20]
763                         [op2] "r" ((int32_t)history[history_idx_2]),
0800137e:   ldr     r3, [r7, #20]
08001380:   lsls    r3, r3, #1
08001382:   ldr     r2, [r7, #0]
08001384:   add     r3, r2
08001386:   ldrsh.w r3, [r3]
0800138a:   mov     r1, r3
764                         [acc1] "r" (accumulators[i])
0800138c:   ldr     r3, [r7, #44]   @ 0x2c
0800138e:   lsls    r3, r3, #2
08001390:   adds    r3, #56 @ 0x38
08001392:   add     r3, r7
08001394:   ldr.w   r2, [r3, #-48]
759                 __asm volatile(
08001398:   ldr     r3, [r7, #36]   @ 0x24
0800139a:   smlabb  r2, r3, r1, r2
```

```
0800139e:   ldr     r3, [r7, #44]   @ 0x2c
080013a0:   lsls    r3, r3, #2
080013a2:   adds    r3, #56 @ 0x38
080013a4:   add     r3, r7
080013a6:   str.w   r2, [r3, #-48]
771                         [op2] "r" ((int32_t)history[history_idx_1]),
080013aa:   ldr     r3, [r7, #24]
080013ac:   lsls    r3, r3, #1
080013ae:   ldr     r2, [r7, #0]
080013b0:   add     r3, r2
080013b2:   ldrsh.w r3, [r3]
080013b6:   mov     r1, r3
772                         [acc2] "r" (accumulators[i])
080013b8:   ldr     r3, [r7, #44]   @ 0x2c
080013ba:   lsls    r3, r3, #2
080013bc:   adds    r3, #56 @ 0x38
080013be:   add     r3, r7
080013c0:   ldr.w   r2, [r3, #-48]
767                   __asm volatile(
080013c4:   ldr     r3, [r7, #32]
080013c6:   smlabb  r2, r3, r1, r2
080013ca:   ldr     r3, [r7, #44]   @ 0x2c
080013cc:   lsls    r3, r3, #2
080013ce:   adds    r3, #56 @ 0x38
080013d0:   add     r3, r7
080013d2:   str.w   r2, [r3, #-48]
754               for (int i = 0; i < FRAME_SIZE; i++) {
080013d6:   ldr     r3, [r7, #44]   @ 0x2c
080013d8:   adds    r3, #1
080013da:   str     r3, [r7, #44]   @ 0x2c
080013dc:   ldr     r3, [r7, #44]   @ 0x2c
080013de:   cmp     r3, #2
080013e0:   ble.n   0x800133c <ProcessBlock2+200>
747             for (tap = 0; tap < (NUMBER_OF_TAPS / 2) * 2; tap += 2, current-=2) {
080013e2:   ldr     r3, [r7, #48]   @ 0x30
080013e4:   adds    r3, #2
080013e6:   str     r3, [r7, #48]   @ 0x30
080013e8:   ldr     r3, [r7, #52]   @ 0x34
080013ea:   subs    r3, #2
080013ec:   str     r3, [r7, #52]   @ 0x34
080013ee:   ldr     r3, [r7, #48]   @ 0x30
080013f0:   cmp     r3, #255        @ 0xff
080013f2:   ble.w   0x80012e0 <ProcessBlock2+108>
797           head = (head + 1) % HISTORY_SIZE;
080013f6:   ldr     r3, [pc, #240]  @ (0x80014e8 <ProcessBlock2+628>)
080013f8:   ldr     r3, [r3, #0]
080013fa:   adds    r1, r3, #1
080013fc:   ldr     r3, [pc, #240]  @ (0x80014f0 <ProcessBlock2+636>)
080013fe:   smull   r2, r3, r3, r1
08001402:   asrs    r2, r3, #7
08001404:   asrs    r3, r1, #31
08001406:   subs    r2, r2, r3
08001408:   mov     r3, r2
0800140a:   lsls    r3, r3, #7
0800140c:   add     r3, r2
0800140e:   lsls    r3, r3, #1
08001410:   add     r3, r2
08001412:   subs    r2, r1, r3
08001414:   ldr     r3, [pc, #208]  @ (0x80014e8 <ProcessBlock2+628>)
08001416:   str     r2, [r3, #0]
800           for (int i = 0; i < FRAME_SIZE; i++) {
08001418:   movs    r3, #0
0800141a:   str     r3, [r7, #40]   @ 0x28
0800141c:   b.n     0x80014d2 <ProcessBlock2+606>
801               if (accumulators[i] > 0x3FFFFFFF) {
0800141e:   ldr     r3, [r7, #40]   @ 0x28
08001420:   lsls    r3, r3, #2
08001422:   adds    r3, #56 @ 0x38
08001424:   add     r3, r7
08001426:   ldr.w   r3, [r3, #-48]
0800142a:   cmp.w   r3, #1073741824 @ 0x40000000
0800142e:   blt.n   0x800144c <ProcessBlock2+472>
802                 accumulators[i] = 0x3FFFFFFF;
08001430:   ldr     r3, [r7, #40]   @ 0x28
08001432:   lsls    r3, r3, #2
08001434:   adds    r3, #56 @ 0x38
08001436:   add     r3, r7
08001438:   mvn.w   r2, #3221225472 @ 0xc0000000
0800143c:   str.w   r2, [r3, #-48]
```

```
803                   overflow_count++;
08001440:   ldr     r3, [pc, #180]  @ (0x80014f8 <ProcessBlock2+644>)
08001442:   ldr     r3, [r3, #0]
08001444:   adds    r3, #1
08001446:   ldr     r2, [pc, #176]  @ (0x80014f8 <ProcessBlock2+644>)
08001448:   str     r3, [r2, #0]
0800144a:   b.n     0x8001478 <ProcessBlock2+516>
804               } else if (accumulators[i] < -0x40000000) {
0800144c:   ldr     r3, [r7, #40]   @ 0x28
0800144e:   lsls    r3, r3, #2
08001450:   adds    r3, #56 @ 0x38
08001452:   add     r3, r7
08001454:   ldr.w   r3, [r3, #-48]
08001458:   cmp.w   r3, #3221225472 @ 0xc0000000
0800145c:   bge.n   0x8001478 <ProcessBlock2+516>
805                   accumulators[i] = -0x40000000;
0800145e:   ldr     r3, [r7, #40]   @ 0x28
08001460:   lsls    r3, r3, #2
08001462:   adds    r3, #56 @ 0x38
08001464:   add     r3, r7
08001466:   mov.w   r2, #3221225472 @ 0xc0000000
0800146a:   str.w   r2, [r3, #-48]
806                   underflow_count++;
0800146e:   ldr     r3, [pc, #140]  @ (0x80014fc <ProcessBlock2+648>)
08001470:   ldr     r3, [r3, #0]
08001472:   adds    r3, #1
08001474:   ldr     r2, [pc, #132]  @ (0x80014fc <ProcessBlock2+648>)
08001476:   str     r3, [r2, #0]
809               if (accumulators[i] < 1000 && accumulators[i] > -1000) {
08001478:   ldr     r3, [r7, #40]   @ 0x28
0800147a:   lsls    r3, r3, #2
0800147c:   adds    r3, #56 @ 0x38
0800147e:   add     r3, r7
08001480:   ldr.w   r3, [r3, #-48]
08001484:   cmp.w   r3, #1000       @ 0x3e8
08001488:   bge.n   0x80014b4 <ProcessBlock2+576>
0800148a:   ldr     r3, [r7, #40]   @ 0x28
0800148c:   lsls    r3, r3, #2
0800148e:   adds    r3, #56 @ 0x38
08001490:   add     r3, r7
08001492:   ldr.w   r3, [r3, #-48]
08001496:   cmn.w   r3, #1000       @ 0x3e8
0800149a:   ble.n   0x80014b4 <ProcessBlock2+576>
810                   accumulators_16[i] = (int16_t)accumulators[i];
0800149c:   ldr     r3, [r7, #40]   @ 0x28
0800149e:   lsls    r3, r3, #2
080014a0:   adds    r3, #56 @ 0x38
080014a2:   add     r3, r7
080014a4:   ldr.w   r3, [r3, #-48]
080014a8:   sxth    r1, r3
080014aa:   ldr     r2, [pc, #84]   @ (0x8001500 <ProcessBlock2+652>)
080014ac:   ldr     r3, [r7, #40]   @ 0x28
080014ae:   strh.w  r1, [r2, r3, lsl #1]
080014b2:   b.n     0x80014cc <ProcessBlock2+600>
812                   accumulators_16[i] = (int16_t)(accumulators[i] >> 15);
080014b4:   ldr     r3, [r7, #40]   @ 0x28
080014b6:   lsls    r3, r3, #2
080014b8:   adds    r3, #56 @ 0x38
080014ba:   add     r3, r7
080014bc:   ldr.w   r3, [r3, #-48]
080014c0:   asrs    r3, r3, #15
080014c2:   sxth    r1, r3
080014c4:   ldr     r2, [pc, #56]   @ (0x8001500 <ProcessBlock2+652>)
080014c6:   ldr     r3, [r7, #40]   @ 0x28
080014c8:   strh.w  r1, [r2, r3, lsl #1]
800             for (int i = 0; i < FRAME_SIZE; i++) {
080014cc:   ldr     r3, [r7, #40]   @ 0x28
080014ce:   adds    r3, #1
080014d0:   str     r3, [r7, #40]   @ 0x28
080014d2:   ldr     r3, [r7, #40]   @ 0x28
080014d4:   cmp     r3, #2
080014d6:   ble.n   0x800141e <ProcessBlock2+426>
816             return true;
080014d8:   movs    r3, #1
817         }
080014da:   mov     r0, r3
080014dc:   adds    r7, #60 @ 0x3c
080014de:   mov     sp, r7
080014e0:   ldr.w   r7, [sp], #4
```

```
080014e4:   bx      lr
```

```
673         static int  ProcessBlock(int16_t newsample, int16_t* history) {
            ProcessBlock:
08001274:   push    {r7, lr}
08001276:   sub     sp, #104        @ 0x68
08001278:   add     r7, sp, #0
0800127a:   mov     r3, r0
0800127c:   str     r1, [r7, #0]
0800127e:   strh    r3, [r7, #6]
675         history[head] = newsample;
08001280:   ldr     r3, [pc, #480]  @ (0x8001464 <ProcessBlock+496>)
08001282:   ldr     r3, [r3, #0]
08001284:   lsls    r3, r3, #1
08001286:   ldr     r2, [r7, #0]
08001288:   add     r3, r2
0800128a:   ldrh    r2, [r7, #6]
0800128c:   strh    r2, [r3, #0]
676         if (samples_since_last_frame++ < FRAME_SIZE - 1 ){
0800128e:   ldr     r3, [pc, #472]  @ (0x8001468 <ProcessBlock+500>)
08001290:   ldr     r3, [r3, #0]
08001292:   adds    r2, r3, #1
08001294:   ldr     r1, [pc, #464]  @ (0x8001468 <ProcessBlock+500>)
08001296:   str     r2, [r1, #0]
08001298:   cmp     r3, #14
0800129a:   bgt.n   0x80012c0 <ProcessBlock+76>
677             head = (head + 1) % HISTORY_SIZE;
0800129c:   ldr     r3, [pc, #452]  @ (0x8001464 <ProcessBlock+496>)
0800129e:   ldr     r3, [r3, #0]
080012a0:   adds    r1, r3, #1
080012a2:   ldr     r3, [pc, #456]  @ (0x800146c <ProcessBlock+504>)
080012a4:   smull   r2, r3, r3, r1
080012a8:   asrs    r2, r3, #7
080012aa:   asrs    r3, r1, #31
080012ac:   subs    r2, r2, r3
080012ae:   mov     r3, r2
080012b0:   lsls    r3, r3, #4
080012b2:   add     r3, r2
080012b4:   lsls    r3, r3, #4
080012b6:   subs    r2, r1, r3
080012b8:   ldr     r3, [pc, #424]  @ (0x8001464 <ProcessBlock+496>)
080012ba:   str     r2, [r3, #0]
679             return false;
080012bc:   movs    r3, #0
080012be:   b.n     0x800145a <ProcessBlock+486>
682         samples_since_last_frame = 0;
080012c0:   ldr     r3, [pc, #420]  @ (0x8001468 <ProcessBlock+500>)
080012c2:   movs    r2, #0
080012c4:   str     r2, [r3, #0]
687         int32_t accumulators [FRAME_SIZE] = {0}; // accumulator[2] corresponds to the newest sample
080012c6:   add.w   r3, r7, #12
080012ca:   movs    r2, #64 @ 0x40
080012cc:   movs    r1, #0
080012ce:   mov     r0, r3
080012d0:   bl      0x8003d34 <memset>
688         for (tap = 0, current = head; tap < NUMBER_OF_TAPS; tap++, current--) {
080012d4:   movs    r3, #0
080012d6:   str     r3, [r7, #100]  @ 0x64
080012d8:   ldr     r3, [pc, #392]  @ (0x8001464 <ProcessBlock+496>)
080012da:   ldr     r3, [r3, #0]
080012dc:   str     r3, [r7, #96]   @ 0x60
080012de:   b.n     0x8001372 <ProcessBlock+254>
689             int32_t coeff = (int32_t)filter_coeffs[tap]; // loading the coefficient once
080012e0:   ldr     r2, [pc, #396]  @ (0x8001470 <ProcessBlock+508>)
080012e2:   ldr     r3, [r7, #100]  @ 0x64
080012e4:   ldrsh.w r3, [r2, r3, lsl #1]
080012e8:   str     r3, [r7, #84]   @ 0x54
691             int base_idx = (current - MAX_FRAME_IDX + HISTORY_SIZE) % HISTORY_SIZE;
080012ea:   ldr     r3, [r7, #96]   @ 0x60
080012ec:   addw    r2, r3, #257    @ 0x101
080012f0:   ldr     r3, [pc, #376]  @ (0x800146c <ProcessBlock+504>)
080012f2:   smull   r1, r3, r3, r2
080012f6:   asrs    r1, r3, #7
080012f8:   asrs    r3, r2, #31
080012fa:   subs    r1, r1, r3
080012fc:   mov     r3, r1
080012fe:   lsls    r3, r3, #4
```

```
08001300:   add     r3, r1
08001302:   lsls    r3, r3, #4
08001304:   subs    r3, r2, r3
08001306:   str     r3, [r7, #80]   @ 0x50
693                 for (int i = 0; i < FRAME_SIZE; i++){
08001308:   movs    r3, #0
0800130a:   str     r3, [r7, #92]   @ 0x5c
0800130c:   b.n     0x8001360 <ProcessBlock+236>
694                     int history_idx = (base_idx + i) % HISTORY_SIZE;
0800130e:   ldr     r2, [r7, #80]   @ 0x50
08001310:   ldr     r3, [r7, #92]   @ 0x5c
08001312:   add     r2, r3
08001314:   ldr     r3, [pc, #340]  @ (0x800146c <ProcessBlock+504>)
08001316:   smull   r1, r3, r3, r2
0800131a:   asrs    r1, r3, #7
0800131c:   asrs    r3, r2, #31
0800131e:   subs    r1, r1, r3
08001320:   mov     r3, r1
08001322:   lsls    r3, r3, #4
08001324:   add     r3, r1
08001326:   lsls    r3, r3, #4
08001328:   subs    r3, r2, r3
0800132a:   str     r3, [r7, #76]   @ 0x4c
695                     accumulators[i] += coeff * (int32_t)history[history_idx];
0800132c:   ldr     r3, [r7, #92]   @ 0x5c
0800132e:   lsls    r3, r3, #2
08001330:   adds    r3, #104        @ 0x68
08001332:   add     r3, r7
08001334:   ldr.w   r2, [r3, #-92]
08001338:   ldr     r3, [r7, #76]   @ 0x4c
0800133a:   lsls    r3, r3, #1
0800133c:   ldr     r1, [r7, #0]
0800133e:   add     r3, r1
08001340:   ldrsh.w r3, [r3]
08001344:   mov     r1, r3
08001346:   ldr     r3, [r7, #84]   @ 0x54
08001348:   mul.w   r3, r1, r3
0800134c:   add     r2, r3
......... ...
693                 for (int i = 0; i < FRAME_SIZE; i++){
0800135a:   ldr     r3, [r7, #92]   @ 0x5c
0800135c:   adds    r3, #1
0800135e:   str     r3, [r7, #92]   @ 0x5c
08001360:   ldr     r3, [r7, #92]   @ 0x5c
08001362:   cmp     r3, #15
08001364:   ble.n   0x800130e <ProcessBlock+154>
688         for (tap = 0, current = head; tap < NUMBER_OF_TAPS; tap++, current--) {
08001366:   ldr     r3, [r7, #100]  @ 0x64
08001368:   adds    r3, #1
0800136a:   str     r3, [r7, #100]  @ 0x64
0800136c:   ldr     r3, [r7, #96]   @ 0x60
0800136e:   subs    r3, #1
08001370:   str     r3, [r7, #96]   @ 0x60
08001372:   ldr     r3, [r7, #100]  @ 0x64
08001374:   cmp     r3, #255        @ 0xff
08001376:   ble.n   0x80012e0 <ProcessBlock+108>
701         head = (head + 1) % HISTORY_SIZE; // moving the head
08001378:   ldr     r3, [pc, #232]  @ (0x8001464 <ProcessBlock+496>)
0800137a:   ldr     r3, [r3, #0]
0800137c:   adds    r1, r3, #1
0800137e:   ldr     r3, [pc, #236]  @ (0x800146c <ProcessBlock+504>)
08001380:   smull   r2, r3, r3, r1
08001384:   asrs    r2, r3, #7
08001386:   asrs    r3, r1, #31
08001388:   subs    r2, r2, r3
0800138a:   mov     r3, r2
0800138c:   lsls    r3, r3, #4
0800138e:   add     r3, r2
08001390:   lsls    r3, r3, #4
08001392:   subs    r2, r1, r3
08001394:   ldr     r3, [pc, #204]  @ (0x8001464 <ProcessBlock+496>)
08001396:   str     r2, [r3, #0]
704         for (int i = 0; i < FRAME_SIZE; i++){
08001398:   movs    r3, #0
0800139a:   str     r3, [r7, #88]   @ 0x58
0800139c:   b.n     0x8001452 <ProcessBlock+478>
705             if (accumulators[i] > 0x3FFFFFFF) {
0800139e:   ldr     r3, [r7, #88]   @ 0x58
080013a0:   lsls    r3, r3, #2
```

```
080013a2:   adds    r3, #104        @ 0x68
080013a4:   add     r3, r7
080013a6:   ldr.w   r3, [r3, #-92]
080013aa:   cmp.w   r3, #1073741824 @ 0x40000000
080013ae:   blt.n   0x80013cc <ProcessBlock+344>
706                     accumulators[i]= 0x3FFFFFFF;
080013b0:   ldr     r3, [r7, #88]   @ 0x58
080013b2:   lsls    r3, r3, #2
080013b4:   adds    r3, #104        @ 0x68
080013b6:   add     r3, r7
080013b8:   mvn.w   r2, #3221225472 @ 0xc0000000
080013bc:   str.w   r2, [r3, #-92]
707                     overflow_count++;
080013c0:   ldr     r3, [pc, #176]  @ (0x8001474 <ProcessBlock+512>)
080013c2:   ldr     r3, [r3, #0]
080013c4:   adds    r3, #1
080013c6:   ldr     r2, [pc, #172]  @ (0x8001474 <ProcessBlock+512>)
080013c8:   str     r3, [r2, #0]
080013ca:   b.n     0x80013f8 <ProcessBlock+388>
708                 } else if (accumulators[i] < -0x40000000) {
080013cc:   ldr     r3, [r7, #88]   @ 0x58
080013ce:   lsls    r3, r3, #2
080013d0:   adds    r3, #104        @ 0x68
080013d2:   add     r3, r7
080013d4:   ldr.w   r3, [r3, #-92]
080013d8:   cmp.w   r3, #3221225472 @ 0xc0000000
080013dc:   bge.n   0x80013f8 <ProcessBlock+388>
709                     accumulators[i] = -0x40000000;
080013de:   ldr     r3, [r7, #88]   @ 0x58
080013e0:   lsls    r3, r3, #2
080013e2:   adds    r3, #104        @ 0x68
080013e4:   add     r3, r7
080013e6:   mov.w   r2, #3221225472 @ 0xc0000000
080013ea:   str.w   r2, [r3, #-92]
710                     underflow_count++;
080013ee:   ldr     r3, [pc, #136]  @ (0x8001478 <ProcessBlock+516>)
080013f0:   ldr     r3, [r3, #0]
080013f2:   adds    r3, #1
080013f4:   ldr     r2, [pc, #128]  @ (0x8001478 <ProcessBlock+516>)
080013f6:   str     r3, [r2, #0]
713             if (accumulators[i] < 1000 && accumulators[i] > -1000) {
080013f8:   ldr     r3, [r7, #88]   @ 0x58
080013fa:   lsls    r3, r3, #2
080013fc:   adds    r3, #104        @ 0x68
080013fe:   add     r3, r7
08001400:   ldr.w   r3, [r3, #-92]
08001404:   cmp.w   r3, #1000       @ 0x3e8
08001408:   bge.n   0x8001434 <ProcessBlock+448>
0800140a:   ldr     r3, [r7, #88]   @ 0x58
0800140c:   lsls    r3, r3, #2
0800140e:   adds    r3, #104        @ 0x68
08001410:   add     r3, r7
08001412:   ldr.w   r3, [r3, #-92]
08001416:   cmn.w   r3, #1000       @ 0x3e8
0800141a:   ble.n   0x8001434 <ProcessBlock+448>
714                 accumulators_16[i] = (int16_t)accumulators[i];
0800141c:   ldr     r3, [r7, #88]   @ 0x58
0800141e:   lsls    r3, r3, #2
08001420:   adds    r3, #104        @ 0x68
08001422:   add     r3, r7
08001424:   ldr.w   r3, [r3, #-92]
08001428:   sxth    r1, r3
0800142a:   ldr     r2, [pc, #80]   @ (0x800147c <ProcessBlock+520>)
0800142c:   ldr     r3, [r7, #88]   @ 0x58
0800142e:   strh.w  r1, [r2, r3, lsl #1]
08001432:   b.n     0x800144c <ProcessBlock+472>
716                 accumulators_16[i]= (int16_t)(accumulators[i] >> 15);
08001434:   ldr     r3, [r7, #88]   @ 0x58
08001436:   lsls    r3, r3, #2
08001438:   adds    r3, #104        @ 0x68
0800143a:   add     r3, r7
0800143c:   ldr.w   r3, [r3, #-92]
08001440:   asrs    r3, r3, #15
08001442:   sxth    r1, r3
08001444:   ldr     r2, [pc, #52]   @ (0x800147c <ProcessBlock+520>)
08001446:   ldr     r3, [r7, #88]   @ 0x58
08001448:   strh.w  r1, [r2, r3, lsl #1]
704         for (int i = 0; i < FRAME_SIZE; i++){
0800144c:   ldr     r3, [r7, #88]   @ 0x58
```

```
0800144e:   adds    r3, #1
08001450:   str     r3, [r7, #88]   @ 0x58
08001452:   ldr     r3, [r7, #88]   @ 0x58
08001454:   cmp     r3, #15
08001456:   ble.n   0x800139e <ProcessBlock+298>
723         return true;
08001458:   movs    r3, #1
726       }
0800145a:   mov     r0, r3
0800145c:   adds    r7, #104        @ 0x68
0800145e:   mov     sp, r7
08001460:   pop     {r7, pc}
```

```
730       static int ProcessBlock2(int16_t newsample, int16_t* history) {
          ProcessBlock2:
08001274:   push    {r7, lr}
08001276:   sub     sp, #112        @ 0x70
08001278:   add     r7, sp, #0
0800127a:   mov     r3, r0
0800127c:   str     r1, [r7, #0]
0800127e:   strh    r3, [r7, #6]
732           history[head] = newsample;
08001280:   ldr     r3, [pc, #596]  @ (0x80014d8 <ProcessBlock2+612>)
08001282:   ldr     r3, [r3, #0]
08001284:   lsls    r3, r3, #1
08001286:   ldr     r2, [r7, #0]
08001288:   add     r3, r2
0800128a:   ldrh    r2, [r7, #6]
0800128c:   strh    r2, [r3, #0]
734           if (samples_since_last_frame++ < FRAME_SIZE - 1) {
0800128e:   ldr     r3, [pc, #588]  @ (0x80014dc <ProcessBlock2+616>)
08001290:   ldr     r3, [r3, #0]
08001292:   adds    r2, r3, #1
08001294:   ldr     r1, [pc, #580]  @ (0x80014dc <ProcessBlock2+616>)
08001296:   str     r2, [r1, #0]
08001298:   cmp     r3, #14
0800129a:   bgt.n   0x80012c0 <ProcessBlock2+76>
735               head = (head + 1) % HISTORY_SIZE;
0800129c:   ldr     r3, [pc, #568]  @ (0x80014d8 <ProcessBlock2+612>)
0800129e:   ldr     r3, [r3, #0]
080012a0:   adds    r1, r3, #1
080012a2:   ldr     r3, [pc, #572]  @ (0x80014e0 <ProcessBlock2+620>)
080012a4:   smull   r2, r3, r3, r1
080012a8:   asrs    r2, r3, #7
080012aa:   asrs    r3, r1, #31
080012ac:   subs    r2, r2, r3
080012ae:   mov     r3, r2
080012b0:   lsls    r3, r3, #4
080012b2:   add     r3, r2
080012b4:   lsls    r3, r3, #4
080012b6:   subs    r2, r1, r3
080012b8:   ldr     r3, [pc, #540]  @ (0x80014d8 <ProcessBlock2+612>)
080012ba:   str     r2, [r3, #0]
736               return false;
080012bc:   movs    r3, #0
080012be:   b.n     0x80014d0 <ProcessBlock2+604>
739           samples_since_last_frame = 0;
080012c0:   ldr     r3, [pc, #536]  @ (0x80014dc <ProcessBlock2+616>)
080012c2:   movs    r2, #0
080012c4:   str     r2, [r3, #0]
742           int current = head;
080012c6:   ldr     r3, [pc, #528]  @ (0x80014d8 <ProcessBlock2+612>)
080012c8:   ldr     r3, [r3, #0]
080012ca:   str     r3, [r7, #108]  @ 0x6c
743           int32_t accumulators[FRAME_SIZE] = {0};  // accumulator[2] corresponds to the newest sample
080012cc:   add.w   r3, r7, #12
080012d0:   movs    r2, #64 @ 0x40
080012d2:   movs    r1, #0
080012d4:   mov     r0, r3
080012d6:   bl      0x8003da8 <memset>
747           for (tap = 0; tap < (NUMBER_OF_TAPS / 2) * 2; tap += 2, current-=2) {
080012da:   movs    r3, #0
080012dc:   str     r3, [r7, #104]  @ 0x68
080012de:   b.n     0x80013e6 <ProcessBlock2+370>
748               int32_t coeff1 = (int32_t)filter_coeffs[tap];     // First tap coefficient
080012e0:   ldr     r2, [pc, #512]  @ (0x80014e4 <ProcessBlock2+624>)
```

```
080012e2:   ldr     r3, [r7, #104]  @ 0x68
080012e4:   ldrsh.w r3, [r2, r3, lsl #1]
080012e8:   str     r3, [r7, #92]   @ 0x5c
749               int32_t coeff2 = (int32_t)filter_coeffs[tap + 1]; // Second tap coefficient
080012ea:   ldr     r3, [r7, #104]  @ 0x68
080012ec:   adds    r3, #1
080012ee:   ldr     r2, [pc, #500]  @ (0x80014e4 <ProcessBlock2+624>)
080012f0:   ldrsh.w r3, [r2, r3, lsl #1]
080012f4:   str     r3, [r7, #88]   @ 0x58
750             current = (current + HISTORY_SIZE) % HISTORY_SIZE;
080012f6:   ldr     r3, [r7, #108]  @ 0x6c
080012f8:   add.w   r2, r3, #272    @ 0x110
080012fc:   ldr     r3, [pc, #480]  @ (0x80014e0 <ProcessBlock2+620>)
080012fe:   smull   r1, r3, r3, r2
08001302:   asrs    r1, r3, #7
08001304:   asrs    r3, r2, #31
08001306:   subs    r1, r1, r3
08001308:   mov     r3, r1
0800130a:   lsls    r3, r3, #4
0800130c:   add     r3, r1
0800130e:   lsls    r3, r3, #4
08001310:   subs    r3, r2, r3
08001312:   str     r3, [r7, #108]  @ 0x6c
752               int base_idx = (current - FRAME_SIZE + HISTORY_SIZE) % HISTORY_SIZE;
08001314:   ldr     r3, [r7, #108]  @ 0x6c
08001316:   add.w   r2, r3, #256    @ 0x100
0800131a:   ldr     r3, [pc, #452]  @ (0x80014e0 <ProcessBlock2+620>)
0800131c:   smull   r1, r3, r3, r2
08001320:   asrs    r1, r3, #7
08001322:   asrs    r3, r2, #31
08001324:   subs    r1, r1, r3
08001326:   mov     r3, r1
08001328:   lsls    r3, r3, #4
0800132a:   add     r3, r1
0800132c:   lsls    r3, r3, #4
0800132e:   subs    r3, r2, r3
08001330:   str     r3, [r7, #84]   @ 0x54
754               for (int i = 0; i < FRAME_SIZE; i++) {
08001332:   movs    r3, #0
08001334:   str     r3, [r7, #100]  @ 0x64
08001336:   b.n     0x80013d4 <ProcessBlock2+352>
755                 int history_idx_1 = (base_idx + i) % HISTORY_SIZE;
08001338:   ldr     r2, [r7, #84]   @ 0x54
0800133a:   ldr     r3, [r7, #100]  @ 0x64
0800133c:   add     r2, r3
0800133e:   ldr     r3, [pc, #416]  @ (0x80014e0 <ProcessBlock2+620>)
08001340:   smull   r1, r3, r3, r2
08001344:   asrs    r1, r3, #7
08001346:   asrs    r3, r2, #31
08001348:   subs    r1, r1, r3
0800134a:   mov     r3, r1
0800134c:   lsls    r3, r3, #4
0800134e:   add     r3, r1
08001350:   lsls    r3, r3, #4
08001352:   subs    r3, r2, r3
08001354:   str     r3, [r7, #80]   @ 0x50
756                 int history_idx_2 = (base_idx + i + 1) % HISTORY_SIZE;
08001356:   ldr     r2, [r7, #84]   @ 0x54
08001358:   ldr     r3, [r7, #100]  @ 0x64
0800135a:   add     r3, r2
0800135c:   adds    r2, r3, #1
0800135e:   ldr     r3, [pc, #384]  @ (0x80014e0 <ProcessBlock2+620>)
08001360:   smull   r1, r3, r3, r2
08001364:   asrs    r1, r3, #7
08001366:   asrs    r3, r2, #31
08001368:   subs    r1, r1, r3
0800136a:   mov     r3, r1
0800136c:   lsls    r3, r3, #4
0800136e:   add     r3, r1
08001370:   lsls    r3, r3, #4
08001372:   subs    r3, r2, r3
08001374:   str     r3, [r7, #76]   @ 0x4c
763                         [op2] "r" ((int32_t)history[history_idx_2]),
08001376:   ldr     r3, [r7, #76]   @ 0x4c
08001378:   lsls    r3, r3, #1
0800137a:   ldr     r2, [r7, #0]
0800137c:   add     r3, r2
0800137e:   ldrsh.w r3, [r3]
08001382:   mov     r1, r3
```

```
764                         [acc1] "r" (accumulators[i])
08001384:   ldr     r3, [r7, #100]  @ 0x64
08001386:   lsls    r3, r3, #2
08001388:   adds    r3, #112        @ 0x70
0800138a:   add     r3, r7
0800138c:   ldr.w   r2, [r3, #-100]
759                   __asm volatile(
08001390:   ldr     r3, [r7, #92]   @ 0x5c
08001392:   smlabb  r2, r3, r1, r2
08001396:   ldr     r3, [r7, #100]  @ 0x64
08001398:   lsls    r3, r3, #2
0800139a:   adds    r3, #112        @ 0x70
0800139c:   add     r3, r7
0800139e:   str.w   r2, [r3, #-100]
771                         [op2] "r" ((int32_t)history[history_idx_1]),
080013a2:   ldr     r3, [r7, #80]   @ 0x50
080013a4:   lsls    r3, r3, #1
080013a6:   ldr     r2, [r7, #0]
080013a8:   add     r3, r2
080013aa:   ldrsh.w r3, [r3]
080013ae:   mov     r1, r3
772                         [acc2] "r" (accumulators[i])
080013b0:   ldr     r3, [r7, #100]  @ 0x64
080013b2:   lsls    r3, r3, #2
080013b4:   adds    r3, #112        @ 0x70
080013b6:   add     r3, r7
080013b8:   ldr.w   r2, [r3, #-100]
767                   __asm volatile(
080013bc:   ldr     r3, [r7, #88]   @ 0x58
080013be:   smlabb  r2, r3, r1, r2
080013c2:   ldr     r3, [r7, #100]  @ 0x64
080013c4:   lsls    r3, r3, #2
080013c6:   adds    r3, #112        @ 0x70
080013c8:   add     r3, r7
080013ca:   str.w   r2, [r3, #-100]
754               for (int i = 0; i < FRAME_SIZE; i++) {
080013ce:   ldr     r3, [r7, #100]  @ 0x64
080013d0:   adds    r3, #1
080013d2:   str     r3, [r7, #100]  @ 0x64
080013d4:   ldr     r3, [r7, #100]  @ 0x64
080013d6:   cmp     r3, #15
080013d8:   ble.n   0x8001338 <ProcessBlock2+196>
747             for (tap = 0; tap < (NUMBER_OF_TAPS / 2) * 2; tap += 2, current-=2) {
080013da:   ldr     r3, [r7, #104]  @ 0x68
080013dc:   adds    r3, #2
080013de:   str     r3, [r7, #104]  @ 0x68
080013e0:   ldr     r3, [r7, #108]  @ 0x6c
080013e2:   subs    r3, #2
080013e4:   str     r3, [r7, #108]  @ 0x6c
080013e6:   ldr     r3, [r7, #104]  @ 0x68
080013e8:   cmp     r3, #255        @ 0xff
080013ea:   ble.w   0x80012e0 <ProcessBlock2+108>
797             head = (head + 1) % HISTORY_SIZE;
080013ee:   ldr     r3, [pc, #232]  @ (0x80014d8 <ProcessBlock2+612>)
080013f0:   ldr     r3, [r3, #0]
080013f2:   adds    r1, r3, #1
080013f4:   ldr     r3, [pc, #232]  @ (0x80014e0 <ProcessBlock2+620>)
080013f6:   smull   r2, r3, r3, r1
080013fa:   asrs    r2, r3, #7
080013fc:   asrs    r3, r1, #31
080013fe:   subs    r2, r2, r3
08001400:   mov     r3, r2
08001402:   lsls    r3, r3, #4
08001404:   add     r3, r2
08001406:   lsls    r3, r3, #4
08001408:   subs    r2, r1, r3
0800140a:   ldr     r3, [pc, #204]  @ (0x80014d8 <ProcessBlock2+612>)
0800140c:   str     r2, [r3, #0]
800             for (int i = 0; i < FRAME_SIZE; i++) {
0800140e:   movs    r3, #0
08001410:   str     r3, [r7, #96]   @ 0x60
08001412:   b.n     0x80014c8 <ProcessBlock2+596>
801               if (accumulators[i] > 0x3FFFFFFF) {
08001414:   ldr     r3, [r7, #96]   @ 0x60
08001416:   lsls    r3, r3, #2
08001418:   adds    r3, #112        @ 0x70
0800141a:   add     r3, r7
0800141c:   ldr.w   r3, [r3, #-100]
08001420:   cmp.w   r3, #1073741824 @ 0x40000000
```

```
08001424:   blt.n   0x8001442 <ProcessBlock2+462>
802                   accumulators[i] = 0x3FFFFFFF;
08001426:   ldr     r3, [r7, #96]   @ 0x60
08001428:   lsls    r3, r3, #2
0800142a:   adds    r3, #112        @ 0x70
0800142c:   add     r3, r7
0800142e:   mvn.w   r2, #3221225472 @ 0xc0000000
08001432:   str.w   r2, [r3, #-100]
803                   overflow_count++;
08001436:   ldr     r3, [pc, #176]  @ (0x80014e8 <ProcessBlock2+628>)
08001438:   ldr     r3, [r3, #0]
0800143a:   adds    r3, #1
0800143c:   ldr     r2, [pc, #168]  @ (0x80014e8 <ProcessBlock2+628>)
0800143e:   str     r3, [r2, #0]
08001440:   b.n     0x800146e <ProcessBlock2+506>
804                 } else if (accumulators[i] < -0x40000000) {
08001442:   ldr     r3, [r7, #96]   @ 0x60
08001444:   lsls    r3, r3, #2
08001446:   adds    r3, #112        @ 0x70
08001448:   add     r3, r7
0800144a:   ldr.w   r3, [r3, #-100]
0800144e:   cmp.w   r3, #3221225472 @ 0xc0000000
08001452:   bge.n   0x800146e <ProcessBlock2+506>
805                   accumulators[i] = -0x40000000;
08001454:   ldr     r3, [r7, #96]   @ 0x60
08001456:   lsls    r3, r3, #2
08001458:   adds    r3, #112        @ 0x70
0800145a:   add     r3, r7
0800145c:   mov.w   r2, #3221225472 @ 0xc0000000
08001460:   str.w   r2, [r3, #-100]
806                   underflow_count++;
08001464:   ldr     r3, [pc, #132]  @ (0x80014ec <ProcessBlock2+632>)
08001466:   ldr     r3, [r3, #0]
08001468:   adds    r3, #1
0800146a:   ldr     r2, [pc, #128]  @ (0x80014ec <ProcessBlock2+632>)
0800146c:   str     r3, [r2, #0]
809               if (accumulators[i] < 1000 && accumulators[i] > -1000) {
0800146e:   ldr     r3, [r7, #96]   @ 0x60
08001470:   lsls    r3, r3, #2
08001472:   adds    r3, #112        @ 0x70
08001474:   add     r3, r7
08001476:   ldr.w   r3, [r3, #-100]
0800147a:   cmp.w   r3, #1000       @ 0x3e8
0800147e:   bge.n   0x80014aa <ProcessBlock2+566>
08001480:   ldr     r3, [r7, #96]   @ 0x60
08001482:   lsls    r3, r3, #2
08001484:   adds    r3, #112        @ 0x70
08001486:   add     r3, r7
08001488:   ldr.w   r3, [r3, #-100]
0800148c:   cmn.w   r3, #1000       @ 0x3e8
08001490:   ble.n   0x80014aa <ProcessBlock2+566>
810                   accumulators_16[i] = (int16_t)accumulators[i];
08001492:   ldr     r3, [r7, #96]   @ 0x60
08001494:   lsls    r3, r3, #2
08001496:   adds    r3, #112        @ 0x70
08001498:   add     r3, r7
0800149a:   ldr.w   r3, [r3, #-100]
0800149e:   sxth    r1, r3
080014a0:   ldr     r2, [pc, #76]   @ (0x80014f0 <ProcessBlock2+636>)
080014a2:   ldr     r3, [r7, #96]   @ 0x60
080014a4:   strh.w  r1, [r2, r3, lsl #1]
080014a8:   b.n     0x80014c2 <ProcessBlock2+590>
812                   accumulators_16[i] = (int16_t)(accumulators[i] >> 15);
080014aa:   ldr     r3, [r7, #96]   @ 0x60
080014ac:   lsls    r3, r3, #2
080014ae:   adds    r3, #112        @ 0x70
080014b0:   add     r3, r7
080014b2:   ldr.w   r3, [r3, #-100]
080014b6:   asrs    r3, r3, #15
080014b8:   sxth    r1, r3
080014ba:   ldr     r2, [pc, #52]   @ (0x80014f0 <ProcessBlock2+636>)
080014bc:   ldr     r3, [r7, #96]   @ 0x60
080014be:   strh.w  r1, [r2, r3, lsl #1]
800           for (int i = 0; i < FRAME_SIZE; i++) {
080014c2:   ldr     r3, [r7, #96]   @ 0x60
080014c4:   adds    r3, #1
080014c6:   str     r3, [r7, #96]   @ 0x60
080014c8:   ldr     r3, [r7, #96]   @ 0x60
080014ca:   cmp     r3, #15
```

```
080014cc:   ble.n   0x8001414 <ProcessBlock2+416>
816           return true;
080014ce:   movs    r3, #1
817       }
080014d0:   mov     r0, r3
080014d2:   adds    r7, #112        @ 0x70
080014d4:   mov     sp, r7
080014d6:   pop     {r7, pc}
```