[Microprocessor Applications]
Lab 3: NEON Programming

Chester Sungchung Park

SoC Design Lab, Konkuk University

Webpage: http://soclab.konkuk.ac.kr

Outline

- ❑ Objectives
- ❑ Description
- ❑ Block diagram
- ❑ Source codes
- ❑ Evaluation

Objectives

- Optimizing a C application using NEON instruction set
- Understanding the corresponding assembly codes
- Programming your own assembly codes using NEON instruction set

Description

- Vector addition

1

```
void add_int (int *pa, int * pb, unsigned int n, int x)
{
  unsigned int i;
  for(i = 0; i < n; i++)
      pa[i] = pb[i] + x;
}
```

2

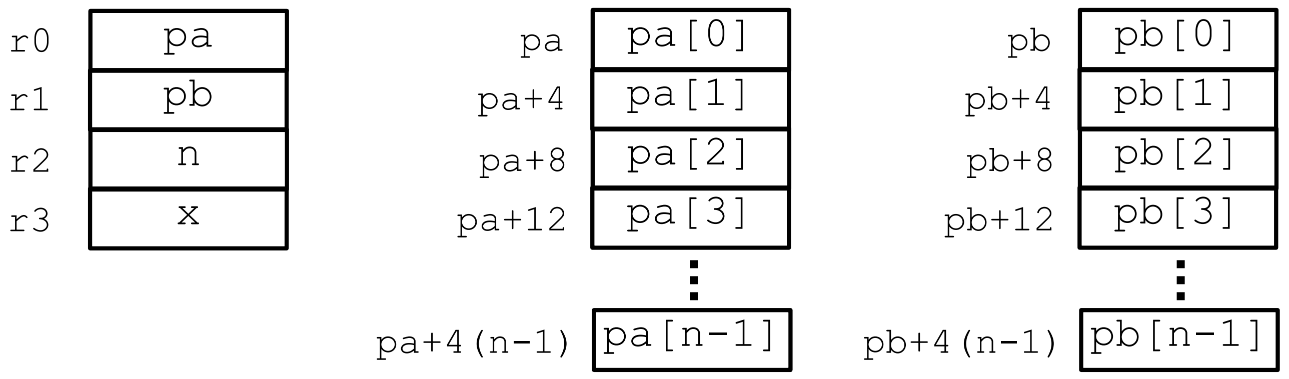
```
void add_int (int *pa, int * pb, unsigned int n, int x)
{
  unsigned int i;
  for(i = 0; i < (n&~3); i++)
      pa[i] = pb[i] + x;
}
```

3

```
void add_int (int* restrict pa, int* restrict pb, unsigned int n, int x)
{
  unsigned int i;
  for(i = 0; i < (n&~3); i++)
      pa[i] = pb[i] + x;
}
```

Description

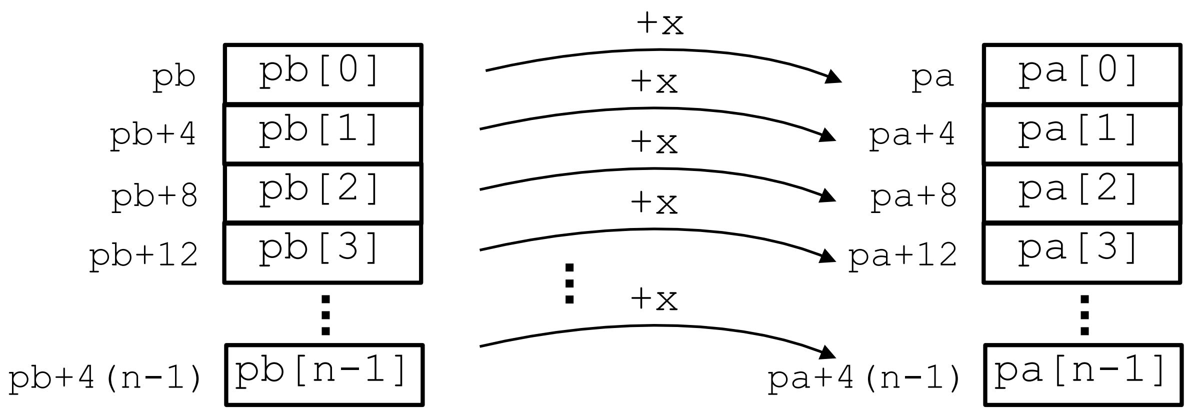
❑ Register/memory setting



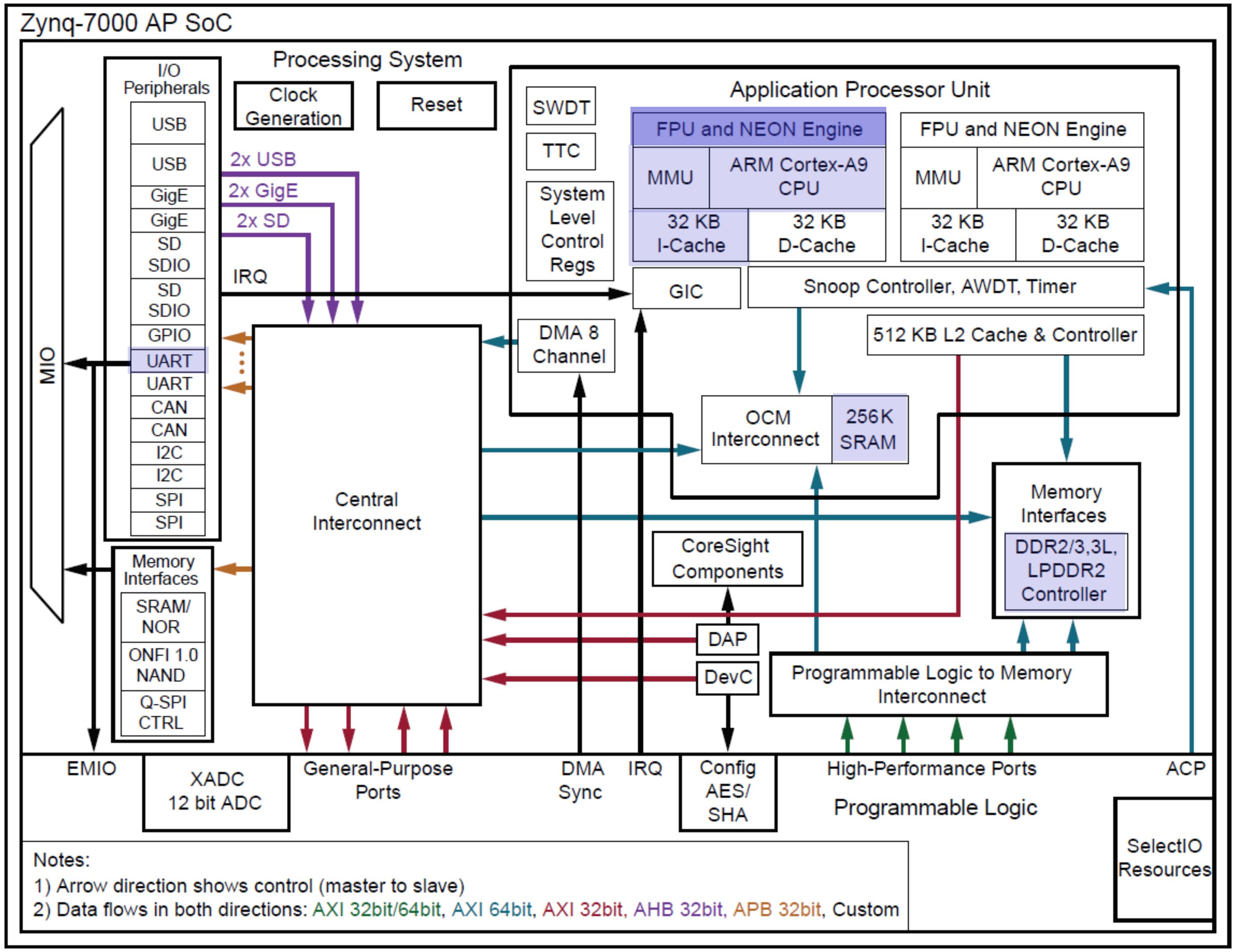
Description

❑ Result

- Non-overlapping memory regions assumed



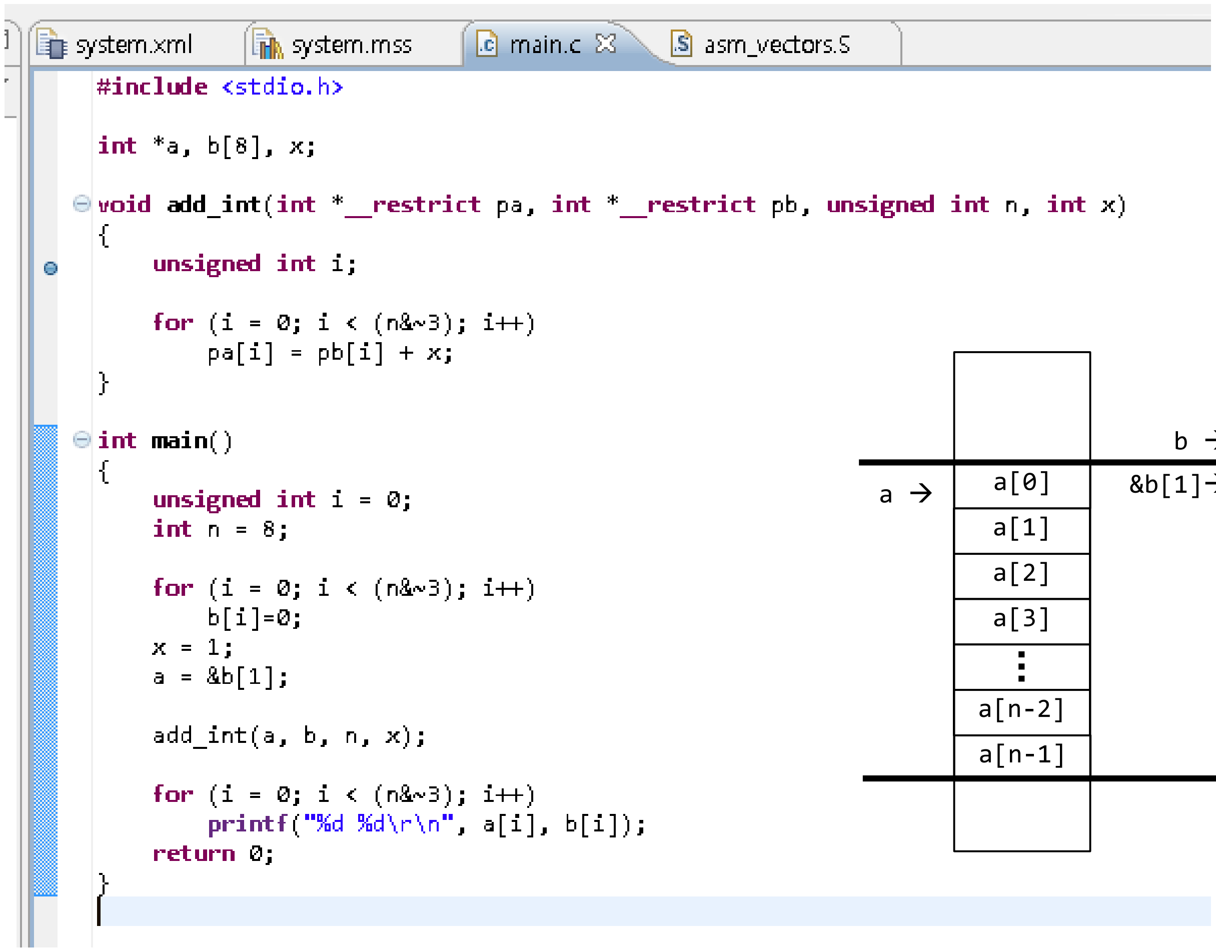
Block Diagram



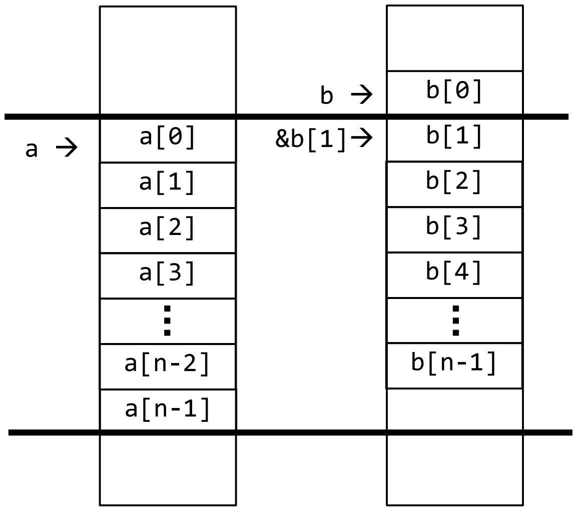
Notes:
1) Arrow direction shows control (master to slave)
2) Data flows in both directions: AXI 32bit/64bit, AXI 64bit, AXI 32bit, AHB 32bit, APB 32bit, Custom

Source Codes

- C program

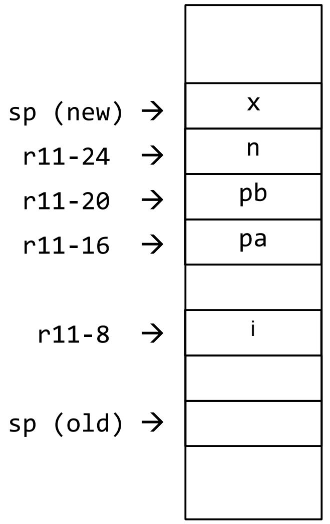


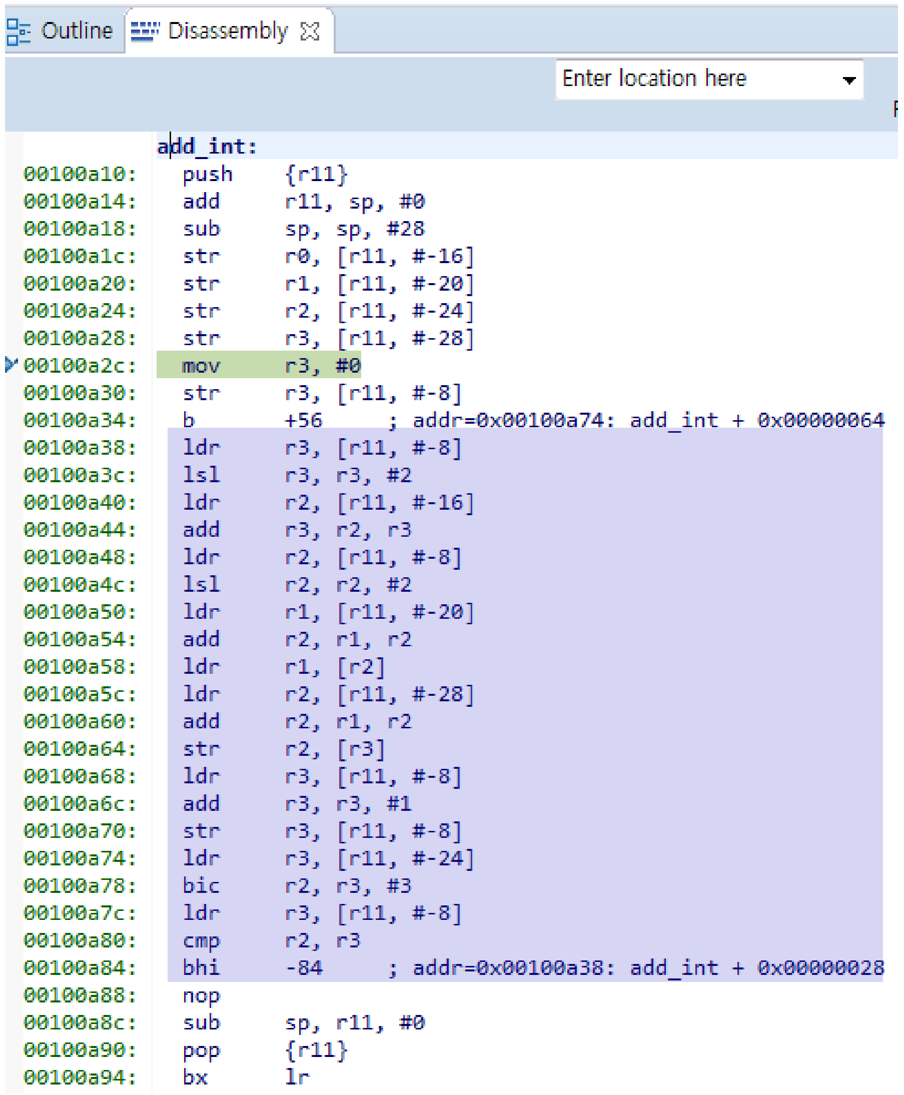
```c
#include <stdio.h>

int *a, b[8], x;

void add_int(int *__restrict pa, int *__restrict pb, unsigned int n, int x)
{
    unsigned int i;

    for (i = 0; i < (n&~3); i++)
        pa[i] = pb[i] + x;
}

int main()
{
    unsigned int i = 0;
    int n = 8;

    for (i = 0; i < (n&~3); i++)
        b[i]=0;
    x = 1;
    a = &b[1];

    add_int(a, b, n, x);

    for (i = 0; i < (n&~3); i++)
        printf("%d %d\r\n", a[i], b[i]);
    return 0;
}
```



Source Codes

❑ Assembly (-O0)

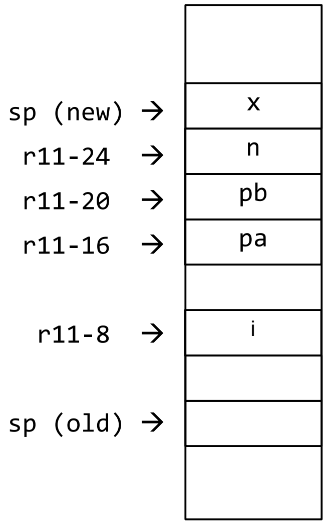


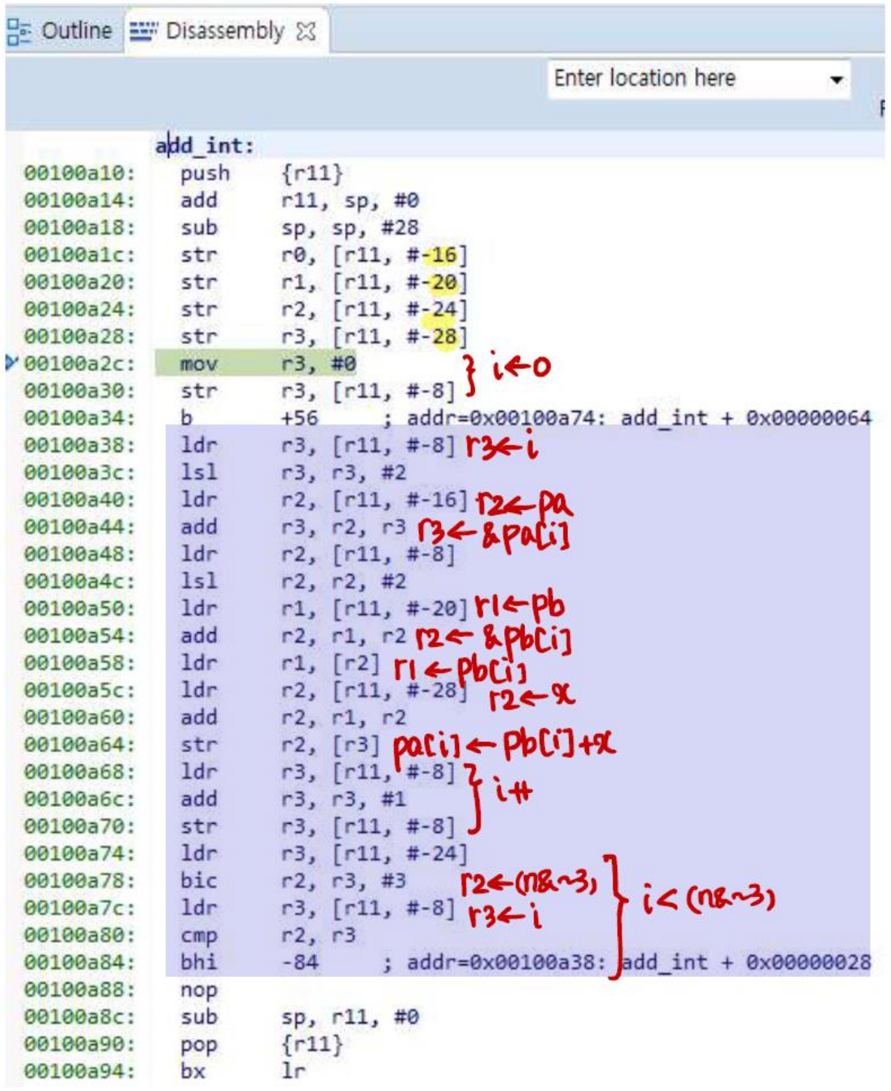


```
add_int:
00100a10:   push    {r11}
00100a14:   add     r11, sp, #0
00100a18:   sub     sp, sp, #28
00100a1c:   str     r0, [r11, #-16]
00100a20:   str     r1, [r11, #-20]
00100a24:   str     r2, [r11, #-24]
00100a28:   str     r3, [r11, #-28]
00100a2c:   mov     r3, #0
00100a30:   str     r3, [r11, #-8]
00100a34:   b       +56       ; addr=0x00100a74: add_int + 0x00000064
00100a38:   ldr     r3, [r11, #-8]
00100a3c:   lsl     r3, r3, #2
00100a40:   ldr     r2, [r11, #-16]
00100a44:   add     r3, r2, r3
00100a48:   ldr     r2, [r11, #-8]
00100a4c:   lsl     r2, r2, #2
00100a50:   ldr     r1, [r11, #-20]
00100a54:   add     r2, r1, r2
00100a58:   ldr     r1, [r2]
00100a5c:   ldr     r2, [r11, #-28]
00100a60:   add     r2, r1, r2
00100a64:   str     r2, [r3]
00100a68:   ldr     r3, [r11, #-8]
00100a6c:   add     r3, r3, #1
00100a70:   str     r3, [r11, #-8]
00100a74:   ldr     r3, [r11, #-24]
00100a78:   bic     r2, r3, #3
00100a7c:   ldr     r3, [r11, #-8]
00100a80:   cmp     r2, r3
00100a84:   bhi     -84       ; addr=0x00100a38: add_int + 0x00000028
00100a88:   nop
00100a8c:   sub     sp, r11, #0
00100a90:   pop     {r11}
00100a94:   bx      lr
```

Source Codes

- ❑ Assembly (-O0)

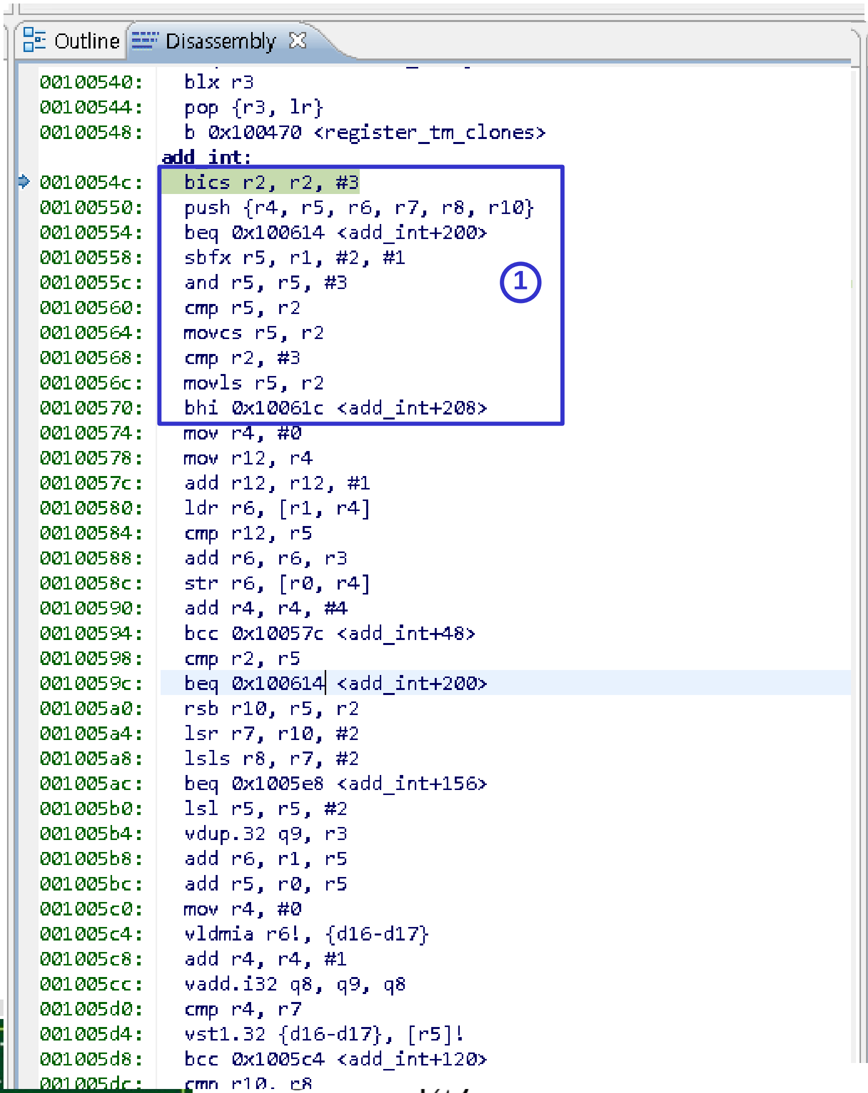




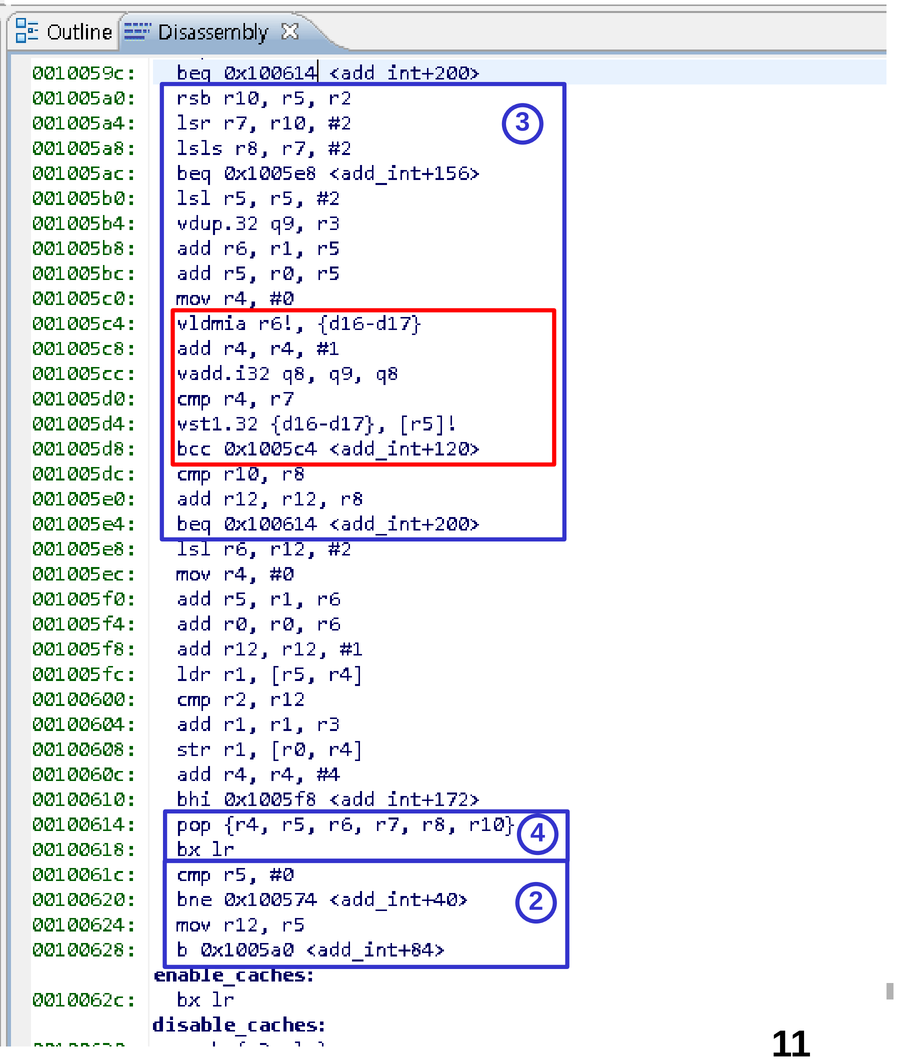
```
          add_int:
00100a10:   push    {r11}
00100a14:   add     r11, sp, #0
00100a18:   sub     sp, sp, #28
00100a1c:   str     r0, [r11, #-16]
00100a20:   str     r1, [r11, #-20]
00100a24:   str     r2, [r11, #-24]
00100a28:   str     r3, [r11, #-28]
00100a2c:   mov     r3, #0            } i←0
00100a30:   str     r3, [r11, #-8]
00100a34:   b       +56       ; addr=0x00100a74: add_int + 0x00000064
00100a38:   ldr     r3, [r11, #-8]  r3←i
00100a3c:   lsl     r3, r3, #2
00100a40:   ldr     r2, [r11, #-16] r2←pa
00100a44:   add     r3, r2, r3  r3←&pa[i]
00100a48:   ldr     r2, [r11, #-8]
00100a4c:   lsl     r2, r2, #2
00100a50:   ldr     r1, [r11, #-20] r1←pb
00100a54:   add     r2, r1, r2  r2←&pb[i]
00100a58:   ldr     r1, [r2]  r1←pb[i]
00100a5c:   ldr     r2, [r11, #-28]  r2←x
00100a60:   add     r2, r1, r2
00100a64:   str     r2, [r3]  pa[i]←pb[i]+x
00100a68:   ldr     r3, [r11, #-8] }
00100a6c:   add     r3, r3, #1     } i++
00100a70:   str     r3, [r11, #-8] }
00100a74:   ldr     r3, [r11, #-24]
00100a78:   bic     r2, r3, #3      r2←(n&~3) }
00100a7c:   ldr     r3, [r11, #-8]  r3←i      } i<(n&~3)
00100a80:   cmp     r2, r3                    }
00100a84:   bhi     -84       ; addr=0x00100a38: add_int + 0x00000028
00100a88:   nop
00100a8c:   sub     sp, r11, #0
00100a90:   pop     {r11}
00100a94:   bx      lr
```

Source Codes

- ❑ Assembly (-O3)



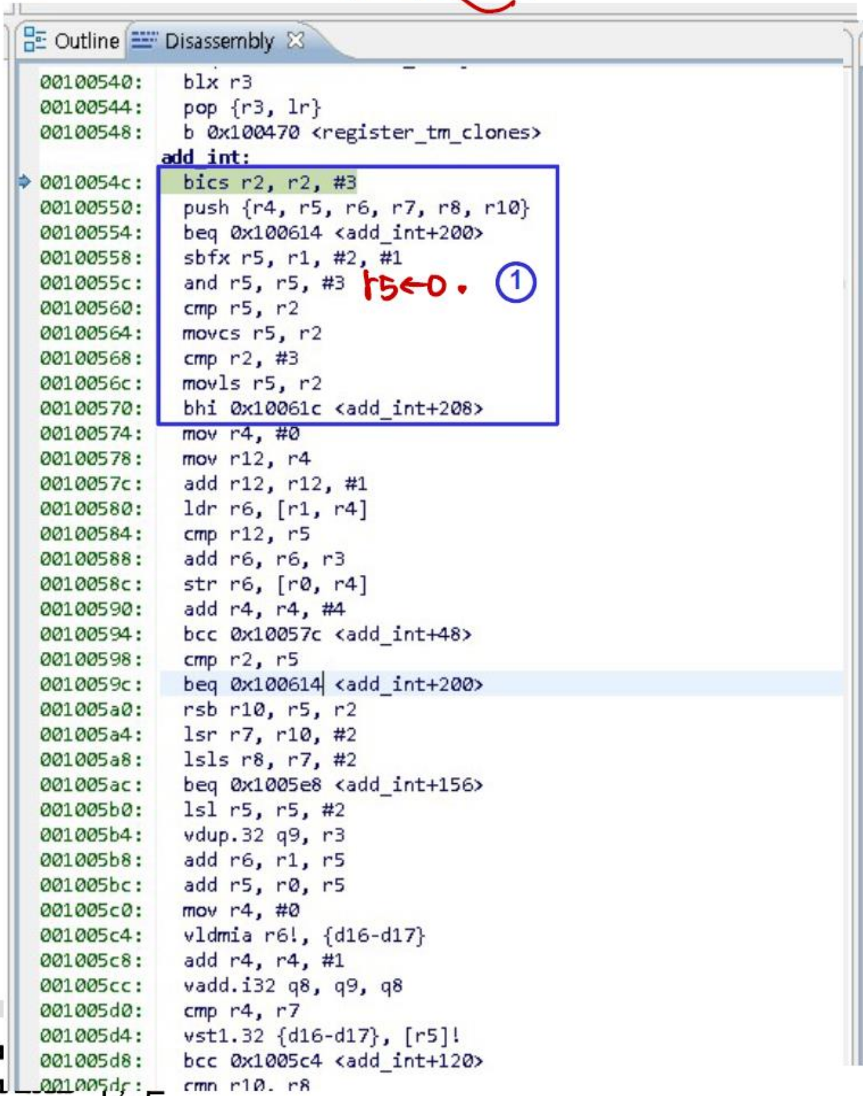
```
00100540:   blx r3
00100544:   pop {r3, lr}
00100548:   b 0x100470 <register_tm_clones>
          add_int:
0010054c:   bics r2, r2, #3
00100550:   push {r4, r5, r6, r7, r8, r10}
00100554:   beq 0x100614 <add_int+200>
00100558:   sbfx r5, r1, #2, #1
0010055c:   and r5, r5, #3
00100560:   cmp r5, r2
00100564:   movcs r5, r2
00100568:   cmp r2, #3
0010056c:   movls r5, r2
00100570:   bhi 0x10061c <add_int+208>
00100574:   mov r4, #0
00100578:   mov r12, r4
0010057c:   add r12, r12, #1
00100580:   ldr r6, [r1, r4]
00100584:   cmp r12, r5
00100588:   add r6, r6, r3
0010058c:   str r6, [r0, r4]
00100590:   add r4, r4, #4
00100594:   bcc 0x10057c <add_int+48>
00100598:   cmp r2, r5
0010059c:   beq 0x100614 <add_int+200>
001005a0:   rsb r10, r5, r2
001005a4:   lsr r7, r10, #2
001005a8:   lsls r8, r7, #2
001005ac:   beq 0x1005e8 <add_int+156>
001005b0:   lsl r5, r5, #2
001005b4:   vdup.32 q9, r3
001005b8:   add r6, r1, r5
001005bc:   add r5, r0, r5
001005c0:   mov r4, #0
001005c4:   vldmia r6!, {d16-d17}
001005c8:   add r4, r4, #1
001005cc:   vadd.i32 q8, q9, q8
001005d0:   cmp r4, r7
001005d4:   vst1.32 {d16-d17}, [r5]!
001005d8:   bcc 0x1005c4 <add_int+120>
001005dc:   cmp r10, r8
```



```
0010059c:   beq 0x100614 <add_int+200>
001005a0:   rsb r10, r5, r2
001005a4:   lsr r7, r10, #2
001005a8:   lsls r8, r7, #2
001005ac:   beq 0x1005e8 <add_int+156>
001005b0:   lsl r5, r5, #2
001005b4:   vdup.32 q9, r3
001005b8:   add r6, r1, r5
001005bc:   add r5, r0, r5
001005c0:   mov r4, #0
001005c4:   vldmia r6!, {d16-d17}
001005c8:   add r4, r4, #1
001005cc:   vadd.i32 q8, q9, q8
001005d0:   cmp r4, r7
001005d4:   vst1.32 {d16-d17}, [r5]!
001005d8:   bcc 0x1005c4 <add_int+120>
001005dc:   cmp r10, r8
001005e0:   add r12, r12, r8
001005e4:   beq 0x100614 <add_int+200>
001005e8:   lsl r6, r12, #2
001005ec:   mov r4, #0
001005f0:   add r5, r1, r6
001005f4:   add r0, r0, r6
001005f8:   add r12, r12, #1
001005fc:   ldr r1, [r5, r4]
00100600:   cmp r2, r12
00100604:   add r1, r1, r3
00100608:   str r1, [r0, r4]
0010060c:   add r4, r4, #4
00100610:   bhi 0x1005f8 <add_int+172>
00100614:   pop {r4, r5, r6, r7, r8, r10}
00100618:   bx lr
0010061c:   cmp r5, #0
00100620:   bne 0x100574 <add_int+40>
00100624:   mov r12, r5
00100628:   b 0x1005a0 <add_int+84>
          enable_caches:
0010062c:   bx lr
          disable_caches:
```

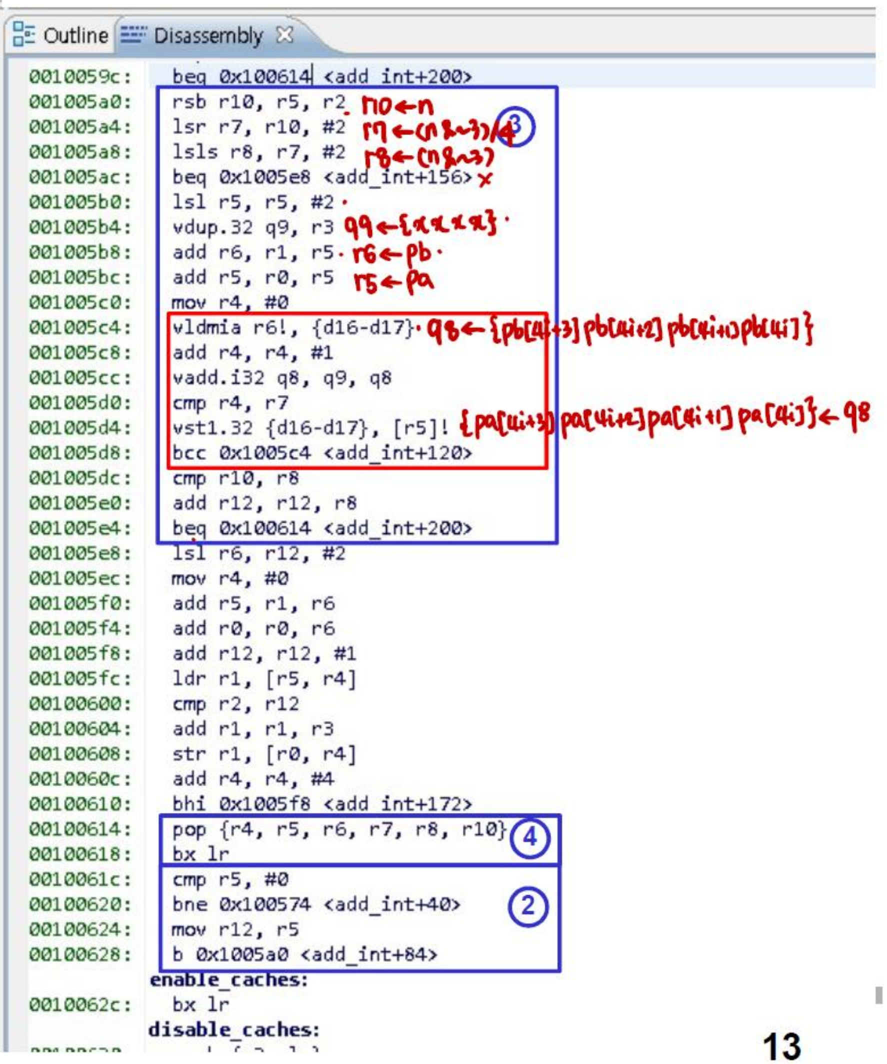
Source Codes

- Assembly (-O3)



Outline | Disassembly

```
00100540:   blx r3
00100544:   pop {r3, lr}
00100548:   b 0x100470 <register_tm_clones>
          add_int:
0010054c:   bics r2, r2, #3
00100550:   push {r4, r5, r6, r7, r8, r10}
00100554:   beq 0x100614 <add_int+200>
00100558:   sbfx r5, r1, #2, #1
0010055c:   and r5, r5, #3     r5←0.    ①
00100560:   cmp r5, r2
00100564:   movcs r5, r2
00100568:   cmp r2, #3
0010056c:   movls r5, r2
00100570:   bhi 0x10061c <add_int+208>
00100574:   mov r4, #0
00100578:   mov r12, r4
0010057c:   add r12, r12, #1
00100580:   ldr r6, [r1, r4]
00100584:   cmp r12, r5
00100588:   add r6, r6, r3
0010058c:   str r6, [r0, r4]
00100590:   add r4, r4, #4
00100594:   bcc 0x10057c <add_int+48>
00100598:   cmp r2, r5
0010059c:   beq 0x100614 <add_int+200>
001005a0:   rsb r10, r5, r2
001005a4:   lsr r7, r10, #2
001005a8:   lsls r8, r7, #2
001005ac:   beq 0x1005e8 <add_int+156>
001005b0:   lsl r5, r5, #2
001005b4:   vdup.32 q9, r3
001005b8:   add r6, r1, r5
001005bc:   add r5, r0, r5
001005c0:   mov r4, #0
001005c4:   vldmia r6!, {d16-d17}
001005c8:   add r4, r4, #1
001005cc:   vadd.i32 q8, q9, q8
001005d0:   cmp r4, r7
001005d4:   vst1.32 {d16-d17}, [r5]!
001005d8:   bcc 0x1005c4 <add_int+120>
001005dc:   cmp r10, r8
```



Outline | Disassembly

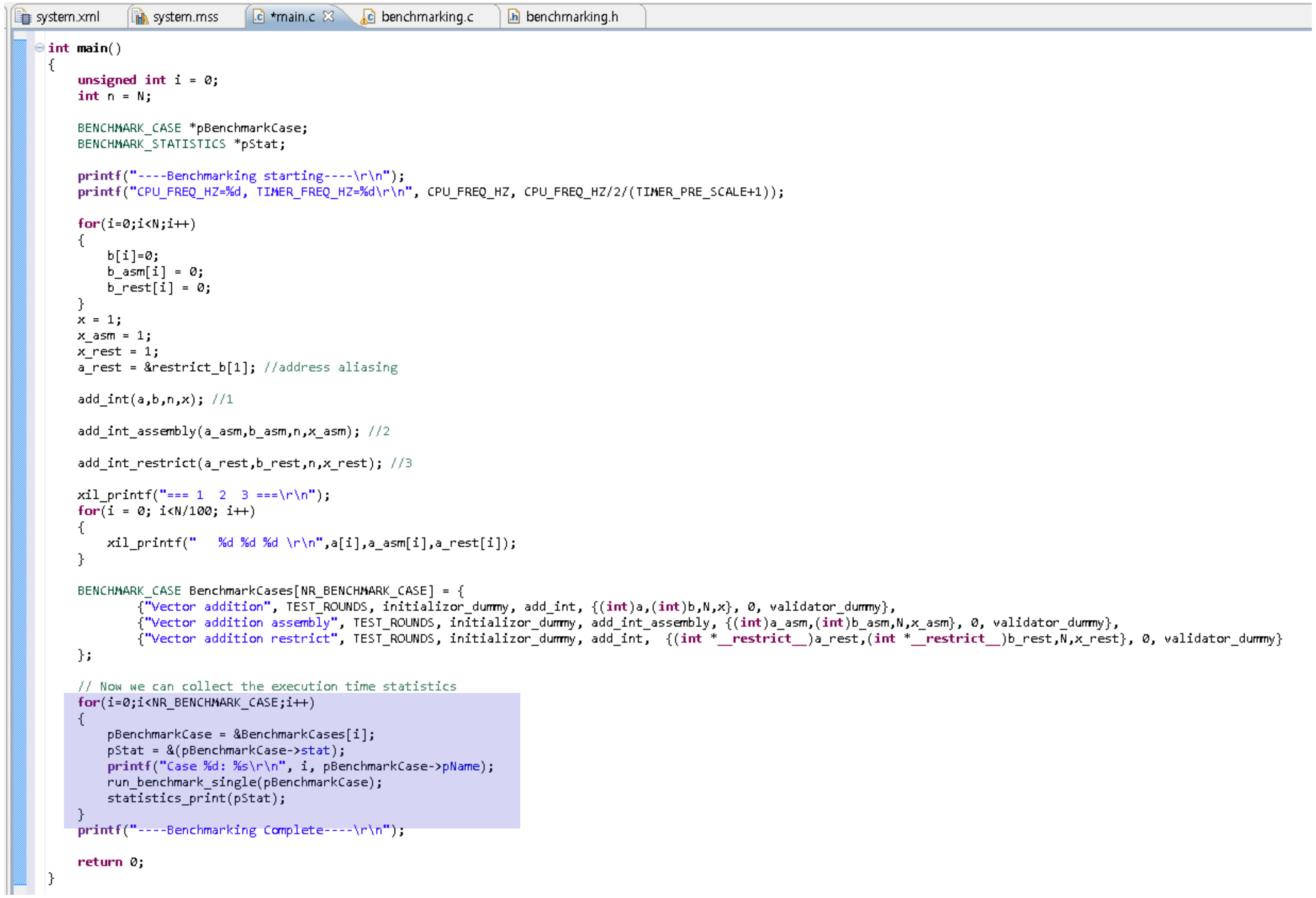
```
0010059c:   beq 0x100614 <add_int+200>
001005a0:   rsb r10, r5, r2.   r10←n
001005a4:   lsr r7, r10, #2    r7←(n&~3)/4   ③
001005a8:   lsls r8, r7, #2    r8←(n&~3)
001005ac:   beq 0x1005e8 <add_int+156> ×
001005b0:   lsl r5, r5, #2 .
001005b4:   vdup.32 q9, r3  q9←{x x x x}.
001005b8:   add r6, r1, r5.  r6←pb.
001005bc:   add r5, r0, r5   r5←pa
001005c0:   mov r4, #0
001005c4:   vldmia r6!, {d16-d17}. q8←{pb[4i+3] pb[4i+2] pb[4i+1] pb[4i]}
001005c8:   add r4, r4, #1
001005cc:   vadd.i32 q8, q9, q8
001005d0:   cmp r4, r7
001005d4:   vst1.32 {d16-d17}, [r5]! {pa[4i+3] pa[4i+2] pa[4i+1] pa[4i]}←q8
001005d8:   bcc 0x1005c4 <add_int+120>
001005dc:   cmp r10, r8
001005e0:   add r12, r12, r8
001005e4:   beq 0x100614 <add_int+200>
001005e8:   lsl r6, r12, #2
001005ec:   mov r4, #0
001005f0:   add r5, r1, r6
001005f4:   add r0, r0, r6
001005f8:   add r12, r12, #1
001005fc:   ldr r1, [r5, r4]
00100600:   cmp r2, r12
00100604:   add r1, r1, r3
00100608:   str r1, [r0, r4]
0010060c:   add r4, r4, #4
00100610:   bhi 0x1005f8 <add_int+172>
00100614:   pop {r4, r5, r6, r7, r8, r10}  ④
00100618:   bx lr
0010061c:   cmp r5, #0
00100620:   bne 0x100574 <add_int+40>     ②
00100624:   mov r12, r5
00100628:   b 0x1005a0 <add_int+84>
          enable_caches:
0010062c:   bx lr
          disable_caches:
```

Evaluation

- Compiler optimization levels (ARM gcc)
  - -O0: No optimization is performed.
  - -O1: Enables the most common forms of optimization that do not require decisions regarding size or speed.
  - -O2: Enables further optimizations, such as instruction scheduling.
  - -O3: Enables more aggressive optimizations, such as aggressive function inlining, and it typically increases speed at the expense of image size. Moreover, this option enables -ftree-vectorize, causing the compiler to attempt to automatically generate NEON code.
  - -Os: Selects optimizations that attempt to minimize the size of the image, even at the expense of speed.

Evaluation

- Source code: main.c

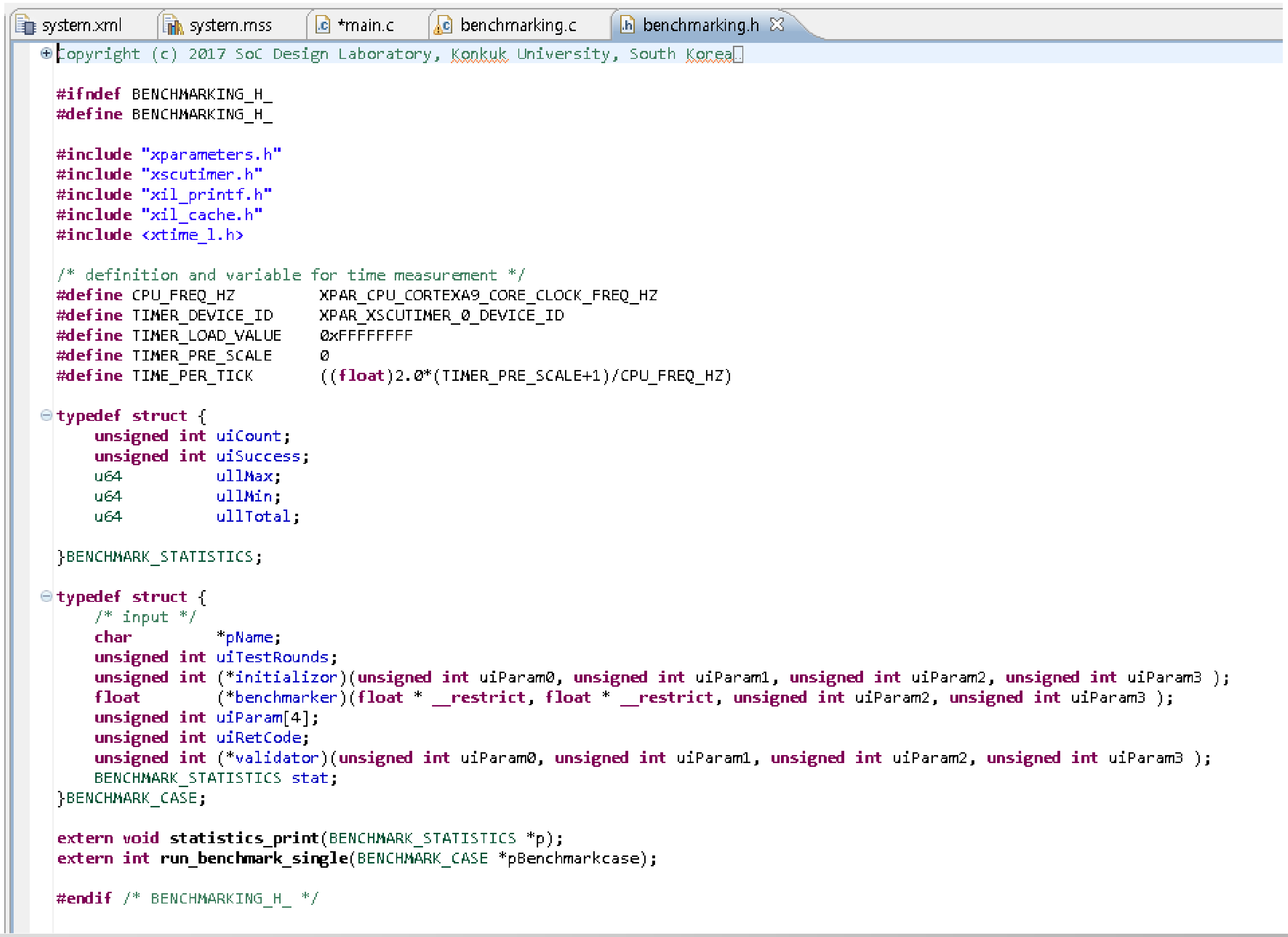


system.xml | system.mss | *main.c | benchmarking.c | benchmarking.h

```
int main()
{
    unsigned int i = 0;
    int n = N;

    BENCHMARK_CASE *pBenchmarkCase;
    BENCHMARK_STATISTICS *pStat;

    printf("----Benchmarking starting----\r\n");
    printf("CPU_FREQ_HZ=%d, TIMER_FREQ_HZ=%d\r\n", CPU_FREQ_HZ, CPU_FREQ_HZ/2/(TIMER_PRE_SCALE+1));

    for(i=0;i<N;i++)
    {
        b[i]=0;
        b_asm[i] = 0;
        b_rest[i] = 0;
    }
    x = 1;
    x_asm = 1;
    x_rest = 1;
    a_rest = &restrict_b[1]; //address aliasing

    add_int(a,b,n,x); //1

    add_int_assembly(a_asm,b_asm,n,x_asm); //2

    add_int_restrict(a_rest,b_rest,n,x_rest); //3

    xil_printf("=== 1  2  3 ===\r\n");
    for(i = 0; i<N/100; i++)
    {
        xil_printf("   %d %d %d \r\n",a[i],a_asm[i],a_rest[i]);
    }

    BENCHMARK_CASE BenchmarkCases[NR_BENCHMARK_CASE] = {
            {"Vector addition", TEST_ROUNDS, initializor_dummy, add_int, {(int)a,(int)b,N,x}, 0, validator_dummy},
            {"Vector addition assembly", TEST_ROUNDS, initializor_dummy, add_int_assembly, {(int)a_asm,(int)b_asm,N,x_asm}, 0, validator_dummy},
            {"Vector addition restrict", TEST_ROUNDS, initializor_dummy, add_int,  {(int *__restrict__)a_rest,(int *__restrict__)b_rest,N,x_rest}, 0, validator_dummy}
    };

    // Now we can collect the execution time statistics
    for(i=0;i<NR_BENCHMARK_CASE;i++)
    {
        pBenchmarkCase = &BenchmarkCases[i];
        pStat = &(pBenchmarkCase->stat);
        printf("Case %d: %s\r\n", i, pBenchmarkCase->pName);
        run_benchmark_single(pBenchmarkCase);
        statistics_print(pStat);
    }
    printf("----Benchmarking Complete----\r\n");

    return 0;
}
```

Evaluation

- Source code: benchmarking.h

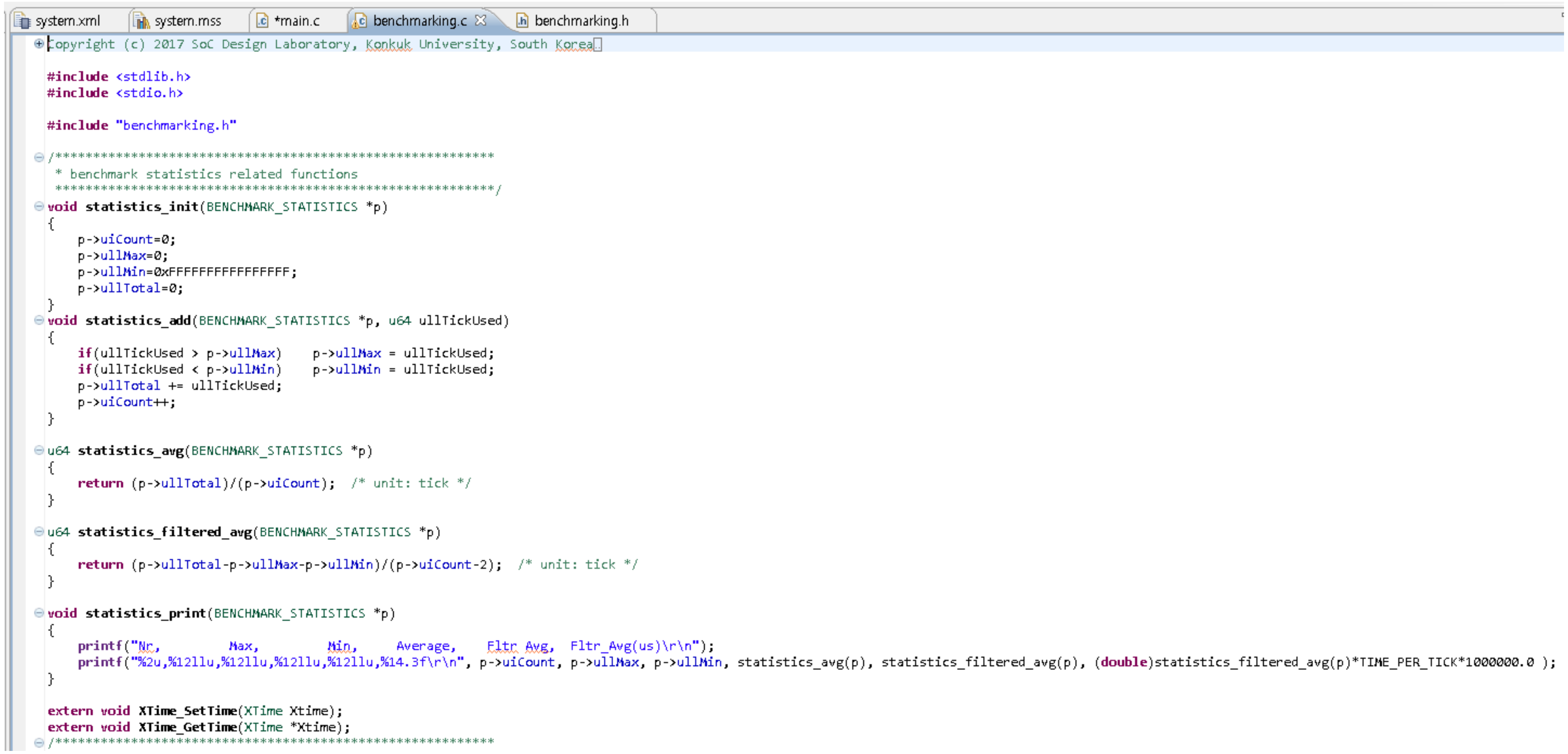


system.xml | system.mss | *main.c | benchmarking.c | benchmarking.h

```c
Copyright (c) 2017 SoC Design Laboratory, Konkuk University, South Korea

#ifndef BENCHMARKING_H_
#define BENCHMARKING_H_

#include "xparameters.h"
#include "xscutimer.h"
#include "xil_printf.h"
#include "xil_cache.h"
#include <xtime_l.h>

/* definition and variable for time measurement */
#define CPU_FREQ_HZ         XPAR_CPU_CORTEXA9_CORE_CLOCK_FREQ_HZ
#define TIMER_DEVICE_ID     XPAR_XSCUTIMER_0_DEVICE_ID
#define TIMER_LOAD_VALUE    0xFFFFFFFF
#define TIMER_PRE_SCALE     0
#define TIME_PER_TICK       ((float)2.0*(TIMER_PRE_SCALE+1)/CPU_FREQ_HZ)

typedef struct {
    unsigned int uiCount;
    unsigned int uiSuccess;
    u64          ullMax;
    u64          ullMin;
    u64          ullTotal;

}BENCHMARK_STATISTICS;

typedef struct {
    /* input */
    char         *pName;
    unsigned int uiTestRounds;
    unsigned int (*initializor)(unsigned int uiParam0, unsigned int uiParam1, unsigned int uiParam2, unsigned int uiParam3 );
    float        (*benchmarker)(float * __restrict, float * __restrict, unsigned int uiParam2, unsigned int uiParam3 );
    unsigned int uiParam[4];
    unsigned int uiRetCode;
    unsigned int (*validator)(unsigned int uiParam0, unsigned int uiParam1, unsigned int uiParam2, unsigned int uiParam3 );
    BENCHMARK_STATISTICS stat;
}BENCHMARK_CASE;

extern void statistics_print(BENCHMARK_STATISTICS *p);
extern int run_benchmark_single(BENCHMARK_CASE *pBenchmarkcase);

#endif /* BENCHMARKING_H_ */
```

Evaluation

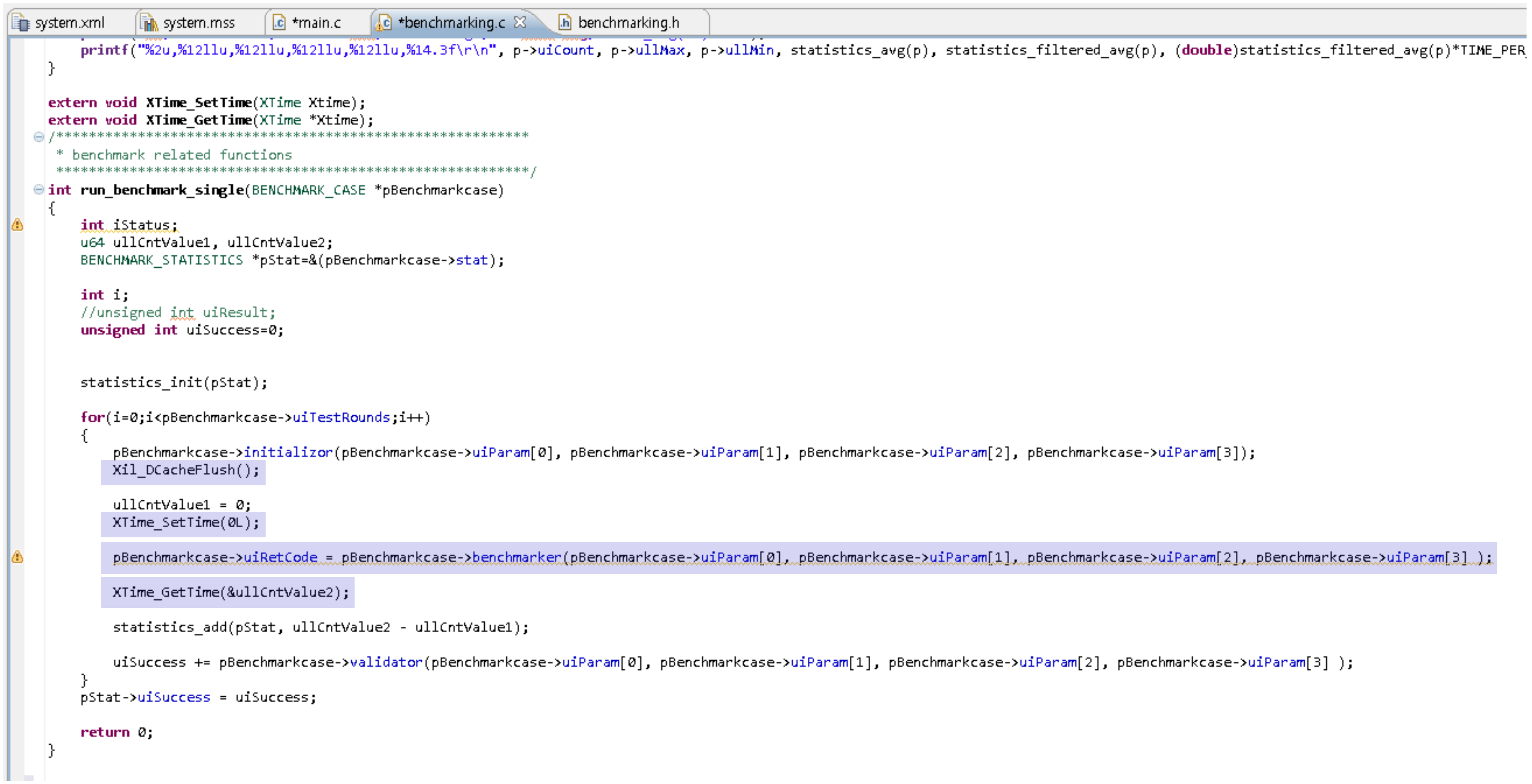
❑ Source code: benchmarking.c



```
Copyright (c) 2017 SoC Design Laboratory, Konkuk University, South Korea

#include <stdlib.h>
#include <stdio.h>

#include "benchmarking.h"

/**********************************************************
 * benchmark statistics related functions
 **********************************************************/
void statistics_init(BENCHMARK_STATISTICS *p)
{
    p->uiCount=0;
    p->ullMax=0;
    p->ullMin=0xFFFFFFFFFFFFFFFF;
    p->ullTotal=0;
}
void statistics_add(BENCHMARK_STATISTICS *p, u64 ullTickUsed)
{
    if(ullTickUsed > p->ullMax)    p->ullMax = ullTickUsed;
    if(ullTickUsed < p->ullMin)    p->ullMin = ullTickUsed;
    p->ullTotal += ullTickUsed;
    p->uiCount++;
}

u64 statistics_avg(BENCHMARK_STATISTICS *p)
{
    return (p->ullTotal)/(p->uiCount);  /* unit: tick */
}

u64 statistics_filtered_avg(BENCHMARK_STATISTICS *p)
{
    return (p->ullTotal-p->ullMax-p->ullMin)/(p->uiCount-2);  /* unit: tick */
}

void statistics_print(BENCHMARK_STATISTICS *p)
{
    printf("Nr,         Max,          Min,      Average,     Fltr Avg,  Fltr_Avg(us)\r\n");
    printf("%2u,%12llu,%12llu,%12llu,%12llu,%14.3f\r\n", p->uiCount, p->ullMax, p->ullMin, statistics_avg(p), statistics_filtered_avg(p), (double)statistics_filtered_avg(p)*TIME_PER_TICK*1000000.0 );
}

extern void XTime_SetTime(XTime Xtime);
extern void XTime_GetTime(XTime *Xtime);
/**********************************************************
```

Evaluation

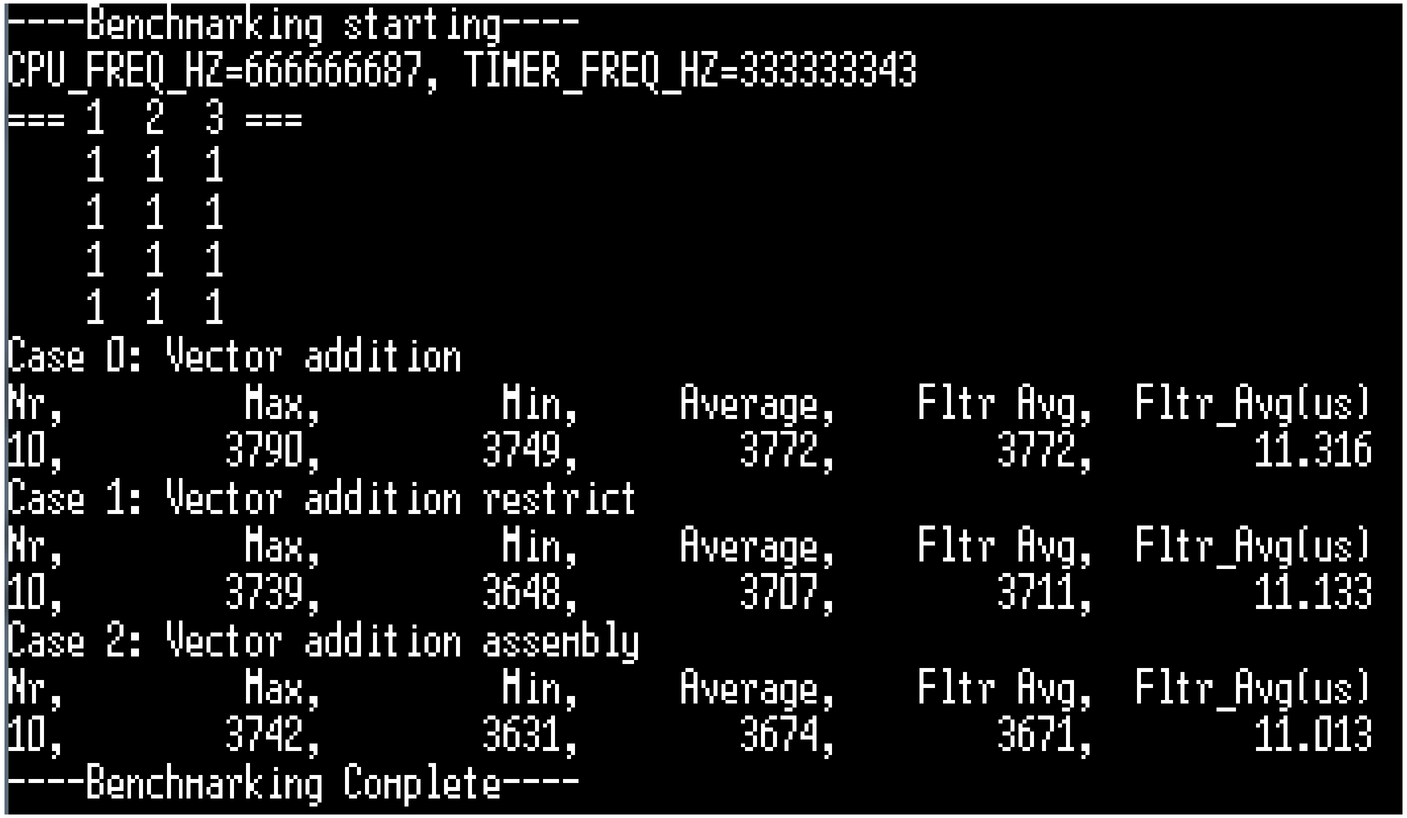
- ❑ Source code: benchmarking.c (cont'd)



```
    printf("%2u,%12llu,%12llu,%12llu,%12llu,%14.3f\r\n", p->uiCount, p->ullMax, p->ullMin, statistics_avg(p), statistics_filtered_avg(p), (double)statistics_filtered_avg(p)*TIME_PER
}

extern void XTime_SetTime(XTime Xtime);
extern void XTime_GetTime(XTime *Xtime);
/*****************************************************
 * benchmark related functions
 *****************************************************/
int run_benchmark_single(BENCHMARK_CASE *pBenchmarkcase)
{
    int iStatus;
    u64 ullCntValue1, ullCntValue2;
    BENCHMARK_STATISTICS *pStat=&(pBenchmarkcase->stat);

    int i;
    //unsigned int uiResult;
    unsigned int uiSuccess=0;


    statistics_init(pStat);

    for(i=0;i<pBenchmarkcase->uiTestRounds;i++)
    {
        pBenchmarkcase->initializor(pBenchmarkcase->uiParam[0], pBenchmarkcase->uiParam[1], pBenchmarkcase->uiParam[2], pBenchmarkcase->uiParam[3]);
        Xil_DCacheFlush();

        ullCntValue1 = 0;
        XTime_SetTime(0L);

        pBenchmarkcase->uiRetCode = pBenchmarkcase->benchmarker(pBenchmarkcase->uiParam[0], pBenchmarkcase->uiParam[1], pBenchmarkcase->uiParam[2], pBenchmarkcase->uiParam[3] );

        XTime_GetTime(&ullCntValue2);

        statistics_add(pStat, ullCntValue2 - ullCntValue1);

        uiSuccess += pBenchmarkcase->validator(pBenchmarkcase->uiParam[0], pBenchmarkcase->uiParam[1], pBenchmarkcase->uiParam[2], pBenchmarkcase->uiParam[3] );
    }
    pStat->uiSuccess = uiSuccess;

    return 0;
}
```

Evaluation

❑ Output in the Console



```
----Benchmarking starting----
CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343
=== 1  2  3 ===
    1  1  1
    1  1  1
    1  1  1
    1  1  1
Case 0: Vector addition
Nr,          Max,          Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,         3790,         3749,         3772,         3772,        11.316
Case 1: Vector addition restrict
Nr,          Max,          Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,         3739,         3648,         3707,         3711,        11.133
Case 2: Vector addition assembly
Nr,          Max,          Min,      Average,     Fltr Avg,  Fltr_Avg(us)
10,         3742,         3631,         3674,         3671,        11.013
----Benchmarking Complete----
```

References

- NEON programmer's guide, version 1.0 (Chapter 3, Appendix C)
- Boost software performance on ZYNQ-7000 AP SoC with NEON, Xilinx, June 2014.