# ECEN 5023-001, -001B, -740

## Mobile Computing & IoT Security

### Lecture #7

### 7 February 2017

Be Boulder.

University of Colorado Boulder

# Agenda

- Class Announcements
- Reading List
- Quiz 4 assigned
- Quiz 3 review
- MCU math 101
- Analog to Digital Converter

# Class Announcements

- Quiz #4 is due at 11:59pm on Sunday, February 12th, 2017
- Self-calibrating ULFRCO assignment is due at 11:59pm on Wednesday, February 8th, 2017

# Reading List

Below is a list of required reading for this course. Questions from these readings plus the lectures from January 17th, 2017 onward will be on the weekly quiz.

1. Circuit Cellar: Electronic Compass: Tilt Compensation & Calibration
   http://cache.nxp.com/files/sensors/doc/reports_presentations/ARTICLE_REPRINT.pdf

2. AN607: Si70XX HUMIDITY AND TEMPERATURE SENSOR DESIGNER'S GUIDE
   https://www.silabs.com/Support%20Documents/TechnicalDocs/AN607.pdf

3. AN580: Infrared Gesture Sensing
   https://www.silabs.com/Support%20Documents/TechnicalDocs/AN580.pdf

Recommended readings. These readings will not be on the weekly quiz, but will be helpful in the class programming assignments and course project.

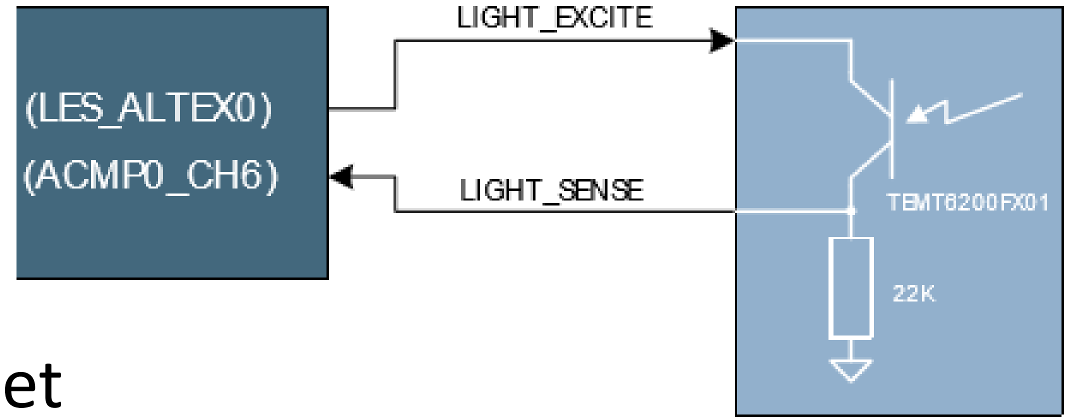
4. AN0013: Direct Memory Access
   http://www.silabs.com/Support%20Documents/TechnicalDocs/AN0013.pdf

5. AN0021: Analog to Digital Converter
   http://www.silabs.com/Support%20Documents/TechnicalDocs/AN0021.pdf

# Quiz 4 assigned

- Quiz #4 is due at 11:59pm on Sunday, February 12th, 2017
- Questions from the required readings plus the lectures from January 17th, 2017 onward will be on the weekly quiz.

# Ambient Light Sensor




- How much current is required to get light_sense to equal 3.3v?
  - 3.3v (light_excite) / 22Kohms = 0.150mA
- Only an estimation since the photo diode is being powered, Vce, to 3.3v, and not 5.0v
  - 800 lx
- Summary:
  - Near dark, light_sense is near 0 volts
  - Approaching full light, light_sense is near 3.3v

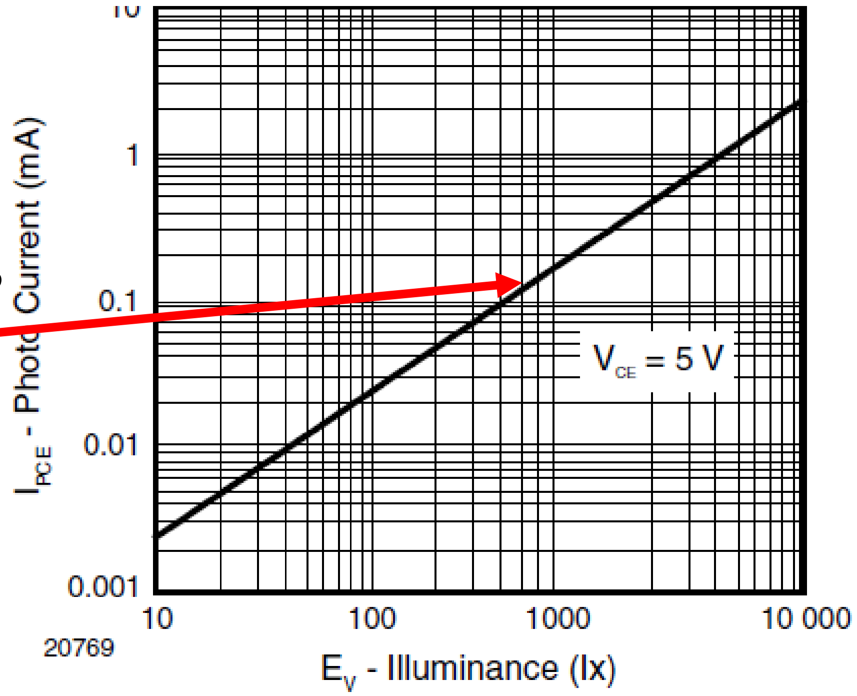



Fig. 4 - Photo Current vs. Illuminance

# Blackboard Review

- When to enable the ACMP0 for lowest energy?

# Blackboard Review

# Blackboard Review

- Calibration theory

# Blackboard Review

# Blackboard Review

- Setting up the calibration on the Leopard Gecko

# Blackboard Review

# Quiz 3 review

Within a Thread network, which network topologies can exist?

- Both Star and Mesh Networks
- Star Network only
- Mesh Network only

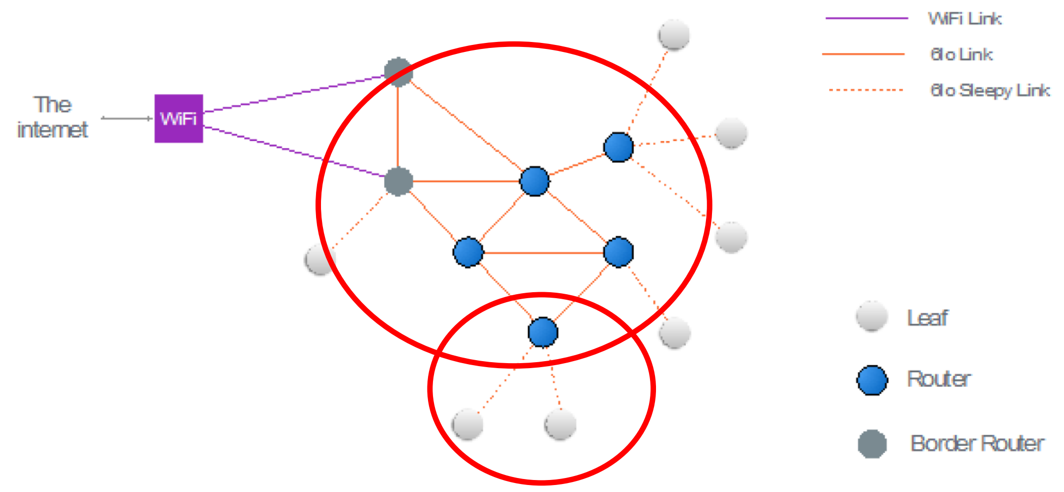



**Figure 3. Basic Thread Network Topology and Devices**

# Quiz 3 review

All Thread device types can enter sleep modes to conserve energy.

- True
- False

# Quiz 3 review

Which is not a characteristic of a Thread Network?

- Low power
- Devices maintain network information upon power loss
- No single point of failure
- Simple network installation
- Designed for small networks only

# Quiz 3 review

In a typical Thread Network, how many bits are used to address the routers?

- 7
- 6
- 10
- 5

# Quiz 3 review

Which is a typical Thread router address?

- 0xC000
- 0xCF00
- 0xDE00
- 0xF010

# Quiz 3 review

In a typical Thread network, which address is not a sleepy end device?

- 0x0C40
- 0xFA01
- 0x0800 ✓
- 0xAA00

# Quiz 3 review

To insure reliable message delivery, Thread's primary method is MAC level retries.

- [x] True
- [ ] False

# Quiz 3 review

To insure reliable message delivery, Thread's primary method is Application level retries.

- ○ True
- ○ False

# Quiz 3 review

What is the order of sequence in joining a Thread Network?

| | |
|---|---|
| 3 | Attaching |
| 2 | Commissioning |
| 1 | Discovery |

# Quiz 3 review

What are the commissioning methods of a device onto a Thread Network? (select all that apply)

- [x] Commissioning session between a joining device and a commissioning application on a smartphone
- [x] Commissioning session between a joining device and a commissioning application on a web browser
- [x] Commission directly on to the device via out-of-band.
- [ ] Push-button joining where there is no user interface or out-of-band channels to device

# Quiz 3 review

A Border Router is a device that provides connectivity of nodes in the Thread Network to other devices in an external network.

- True
- False

# Quiz 3 review

There are a number of variables that impact the actual power consumption during the sleep end point wake sleep cycle. Select all that are implementation specific.

- [x] Time for the device to wake from deep sleep and be ready with the radio
- [x] Active currents during the various operations
- [ ] Time waiting for the ACK packet
- [ ] Time on the air transmitting (size of packet)

# Quiz 3 review

In a typical Thread Network, how many bits are used to address the children?

- 6
- 10
- 5
- 7

# Quiz 3 review

Based on the Leopard Gecko data sheet and reference manual, which energy mode would be specified while the ACMP peripheral is operational and is measuring its input by a call to the blockSleepMode() routine?

- EM0
- EM1
- EM2
- EM3
- EM4

# Quiz 3 review

For the Leopard Gecko STK3600, which instruction would be used to set the gpio pin to exite the on board ambient light sensor?

- GPIO_PinModSet(gpioPortE, 3, gpioModePushPull, 1);
- ✓ GPIO_PinModeSet(gpioPortD, 6, gpioModePushPull, 1);
- GPIO_PinModeSet(gpioPortE, 2, gpioModePushPull, 1);
- GPIO_PinModeSet(gpioPortC, 7, gpioModePushPull, 1);

# Quiz 3 review

To disable Full Bias on the Analog Comparator 0 of the Leopard Gecko peripheral, complete the following c line of code.

ACMP0->CTRL &=

(~ACMP_CTRL_FULLBIAS;, 0x7fff;, 0x7FFF;, ~0x8000;)

# Quiz 3 review

Complete the following line of c code for the Leopard Gecko to read the status of the Leopard Gecko Analog Comparator 1;

int AcmpStatus;

AcmpStatus =

(ACMP1->STATUS;, ACMP1 -> STATUS;, ACMP1-> STATUS;, ACMP1 ->STATUS;)

# Quiz 3 review

If the maximum count of the LETIMER0 is 65,536, what would the LETIMER0 LFXO prescaler need to be to enable the LETIMER0 to count to 18 seconds based on the LFXO set to the frequency of 32,768 to interrupt on the underflow condition only when 18 seconds have past.

(4, four, div16, 16)

What count would be required to be stored in the LETIMER0->CNT register to equal 18 seconds based on the above prescaler which is indicated upon an underflow event?

(36864, 36,864, 36863, 36,863)

# Quiz 3 review

While debugging the following routine, it was determined a line of code is missing. Using the line numbers to the left, at which line should the missing cc

ode be added based on good interrupt handling techniques for non re-entrant interrupt handlers?

```
1.  void LETIMER0_IRQHandler(void) {
2.
3.     int int_flags;
4.
5.     INT_Disable();
6.
7.     int_flags = LETIMER0->IF;
8.
9.     LETIMER0->IFC = int_flags;
10.
11.    if (GPIO_PinOutGet(LED_Port, LED_Pin)) {
12.
13.        GPIO_PinOutClear(LED_Port, LED_Pin); }
14.
15.    else GPIO_PinOutSet(LED_Port, LED_Pint);
16.
17. }
```

(16, sixteen) Write the proper c-code instruction of the missing code? (do not add the line item to the c-code instruction.

The numbers are for reference only.) (INT_Enable();)

# Quiz 3 review

While debugging the following routine, it was determined a line of code is incorrect. Using the line numbers to the left, which line of code is incorrect?

```
1.  void ACMP0_UpdateThreshold(unsigned int VddLevelNew) {
2.
3.    if (ACMP->STATUS) {
4.
5.        ACMP0->INPUTSEL |= VddLevelNew; }
6.
7.    else {
8.
9.        ACMP0->CTRL &= ~ACMP_CTRL_EN; }
10.
11.   }
```

(3, three)

Write the proper c-code instruction of the incorrect line of code? (do not add the line item to the c-code instruction. The numbers are for reference only.)

(3. if (ACMP0->STATUS) {, if (ACMP0->STATUS) {, if (ACMP0->STATUS){, if(ACMP0->STATUS){)

# MCU math 101

- Int a;
- Are these two expression equivalent in c-code? NO

```
a = (5/2) * 4;        a = 5 * (4/2);
a = (2) * 4;          a = 5 * 2;
a = 8;                a = 10;
```

Integer Math

# MCU math 101

- Another example
  - Int Numerator = 50;
  - Int Denominator = 40;
  - Int Results;
    - Results = (Numerator / Denominator) * 100;
    - ~~Results = 120~~
    - ~~Results = 100~~
  - To insure you get the desired results, use the following code:
    - Results = ((float) Numerator / (float) Denominator) * 100;
    - Results = 120
      - The 120 is still stored as an integer into Results

# Cortex-M3: Integer vs floating point addition

C-code for integer addition:

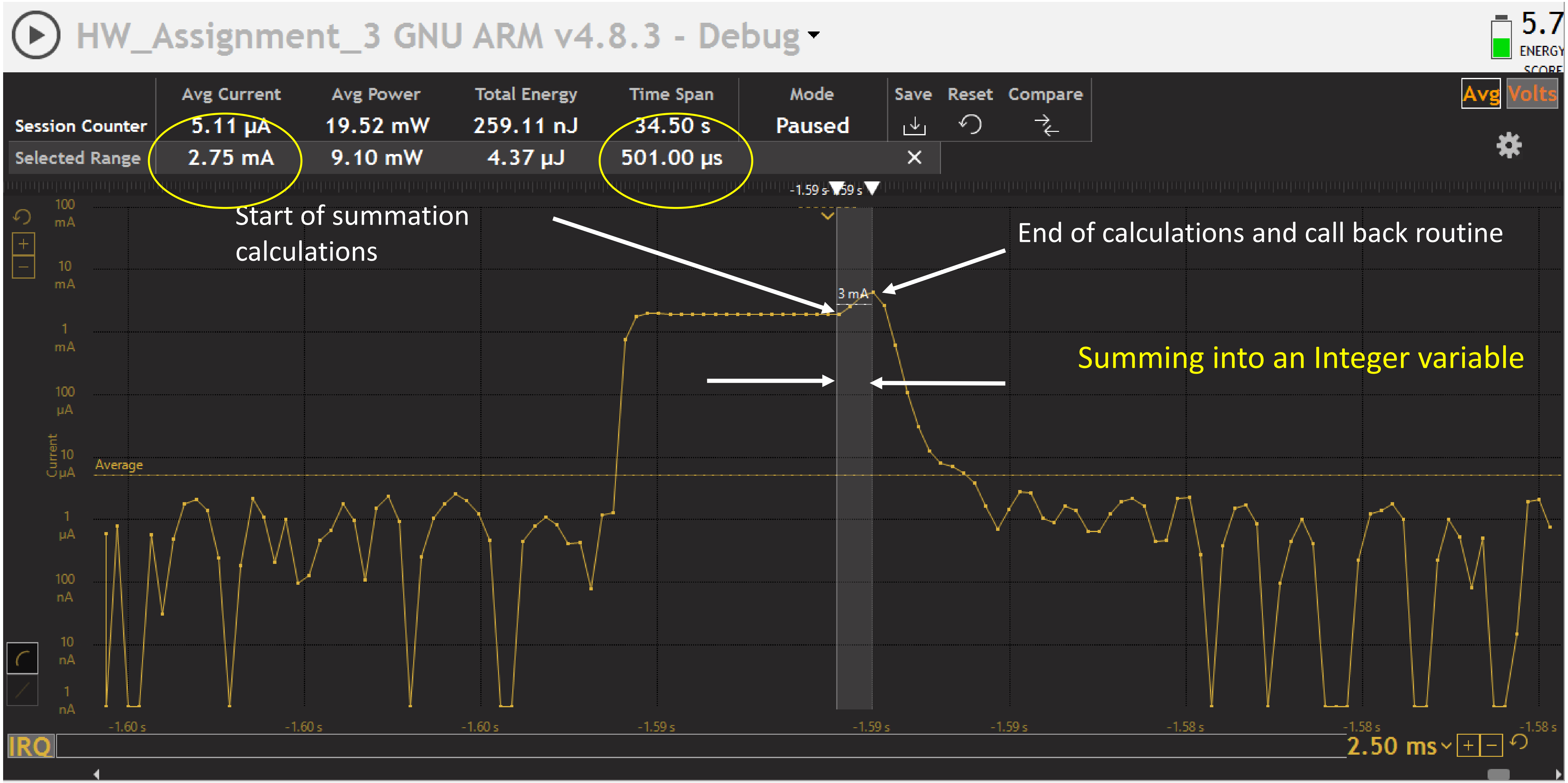
```
int ramBufferData[BufferSize];
int Summation;

Summation = 0;
for(j=0;j<BufferSize;j++) {
    Summation = Summation + ramBufferData[j];
}
```

Assembly code equivalent:

```
Summation = Summation + ramBufferData[j];
00005450:   ldr     r3,[pc,#0x7c] ; 0x54cc
00005452:   ldr     r2,[sp,#0x1c]
00005454:   ldrh.w  r3,[r3,r2,lsl #1]
00005458:   ldr     r2,[sp,#0x18]
0000545a:   add     r3,r2
0000545c:   str     r3,[sp,#0x18]
```

6 Assembly Instructions


HW_Assignment_3 GNU ARM v4.8.3 - Debug
5.7
ENERGY SCORE
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Avg
Volts
Session Counter
5.11 µA
19.52 mW
259.11 nJ
34.50 s
Paused
Selected Range
2.75 mA
9.10 mW
4.37 µJ
501.00 µs
Start of summation calculations
End of calculations and call back routine
Summing into an Integer variable
3 mA
Average
Current
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
1 nA
-1.60 s
-1.59 s
-1.58 s
IRQ
2.50 ms

# 88 Assembly Instructions!

# Cortex-M3: Integer vs floating point addition

C-code for float addition:

```
int ramBufferData[BufferSize];
float Summation;

Summation = 0;
for(j=0;j<BufferSize;j++) {
    Summation = Summation + ramBufferData[j];
}
```

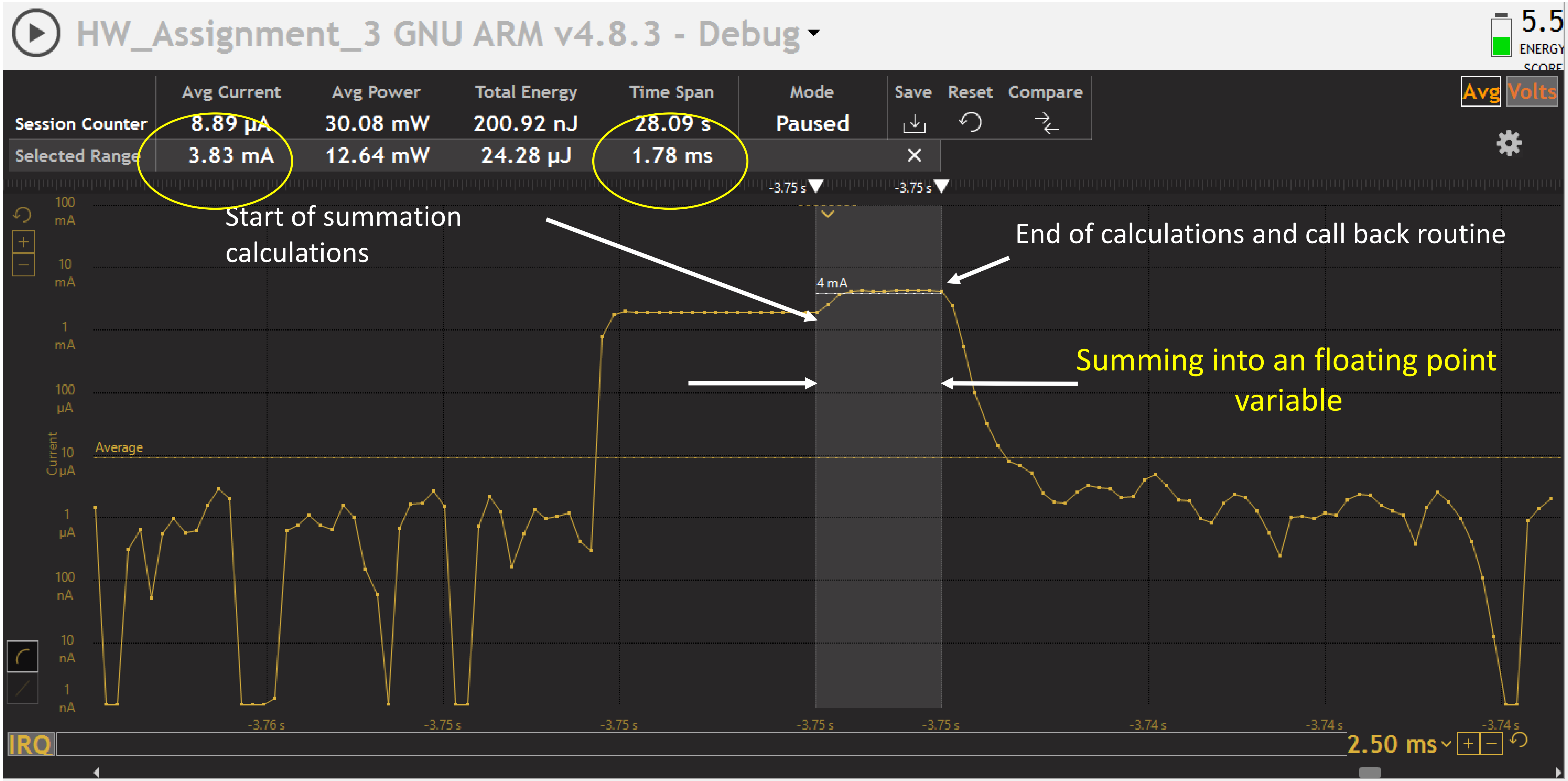
Assembly code equivalent:

```
Summation = Summation + ramBufferAdcData[j];
00005452:  ldr    r3,[pc,#0x90] ; 0x54e0
00005454:  ldr    r2,[sp,#0x1c]
00005456:  ldrh.w  r3,[r3,r2,lsl #1]
0000545a:  mov    r0,r3
0000545c:  bl     0x00005804
00005460:  mov    r3,r0
00005462:  ldr    r0,[sp,#0x18]
00005464:  mov    r1,r3
00005466:  bl     0x0000569c
0000546a:  mov    r3,r0
0000546c:  str    r3,[sp,#0x18]
```

```
00005804:  ands   r3,r0,#0x80000000
00005808:  it     mi
0000580a:  rsbs   r0,r0,#0
0000580c:  movs.w  r12,r0
00005810:  it     eq
00005812:  bx     lr
00005814:  orr    r3,r3,#0x4b000000
00005818:  mov    r1,r0
0000581a:  mov.w  r0,#0x0
0000581e:  b      0x000000000000585a
0000585a:  sub.w  r3,r3,#0x800000
0000585e:  clz    r2,r12
00005862:  subs   r2,#0x8
00005864:  sub.w  r3,r3,r2,lsl #23
00005868:  blt    0x000000000000588c
0000586a:  lsl.w  r12,r1,r2
0000586e:  add    r3,r12
00005870:  lsl.w  r12,r0,r2
00005874:  rsb.w  r2,r2,#0x20
00005878:  cmp.w  r12,#0x80000000
0000587c:  lsr.w  r2,r0,r2
00005880:  adc.w  r0,r3,r2
00005884:  it     eq
00005886:  bic    r0,r0,#0x1
0000588a:  bx     lr
```

```
0000569c:  lsls   r2,r0,#1
0000569e:  itttt  ne
000056a0:  lsls.w  r3,r1,#1
000056a4:  teq    r2,r3
000056a8:  mvns.w  r12,r2,asr #24
000056ac:  mvns.w  r12,r3,asr #24
000056b0:  beq    0x0000000000005788
000056b2:  lsr.w  r2,r2,#24
000056b6:  rsbs   r3,r2,r3,lsr #24
000056ba:  itttt  gt
000056bc:  adds   r2,r2,r3
000056be:  eors   r1,r0
000056c0:  eors   r0,r1
000056c2:  eors   r1,r0
000056c4:  it     lt
000056c6:  rsbs   r3,r3,#0
000056c8:  cmp    r3,#0x19
000056ca:  it     hi
000056cc:  bx     lr
000056ce:  tst    r0,#0x80000000
000056d2:  orr    r0,r0,#0x800000
000056d6:  bic    r0,r0,#0xff000000
000056da:  it     ne
000056dc:  rsbs   r0,r0,#0
000056de:  tst    r1,#0x80000000
000056e2:  orr    r1,r1,#0x800000
000056e6:  bic    r1,r1,#0xff000000
000056ea:  it     ne
000056ec:  rsbs   r1,r1,#0
000056ee:  teq    r2,r3
```

```
000056f2:  beq    0x0000000000005774
000056f4:  sub.w  r2,r2,#0x1
000056f8:  asr.w  r12,r1,r3
000056fc:  adds.w  r0,r0,r12
00005700:  rsb.w  r3,r3,#0x20
00005704:  lsl.w  r1,r1,r3
00005708:  and    r3,r0,#0x80000000
0000570c:  bpl    0x0000000000005714
0000570e:  rsbs   r1,r1,#0
00005710:  sbc.w  r0,r0,r0,lsl #1
00005714:  cmp.w  r0,#0x800000
00005718:  bcc    0x0000000000005742
0000571a:  cmp.w  r0,#0x1000000
0000571e:  bcc    0x000000000000572e
0000572e:  cmp.w  r1,#0x80000000
00005732:  adc.w  r0,r0,r2,lsl #23
00005736:  it     eq
00005738:  bic    r0,r0,#0x1
0000573c:  orr.w  r0,r0,r3
00005740:  bx     lr
```






Floating point summation/averaging of 200 ADC samples

# Energy loss due to use of floating point summation

- Power based from the Energy Profile for DMA call back routine
  - Energy = Power * Time
  - Integer format addition
    - Energy = (3.3v * 2.75mA) * 0.501mS
    - Energy = 4.55 uJ
  - Floating format addition
    - Energy = (3.3v * 3.83mA) * 1.78mS
    - Energy = 22.5uJ

Integer addition is ~ 5X more energy efficient

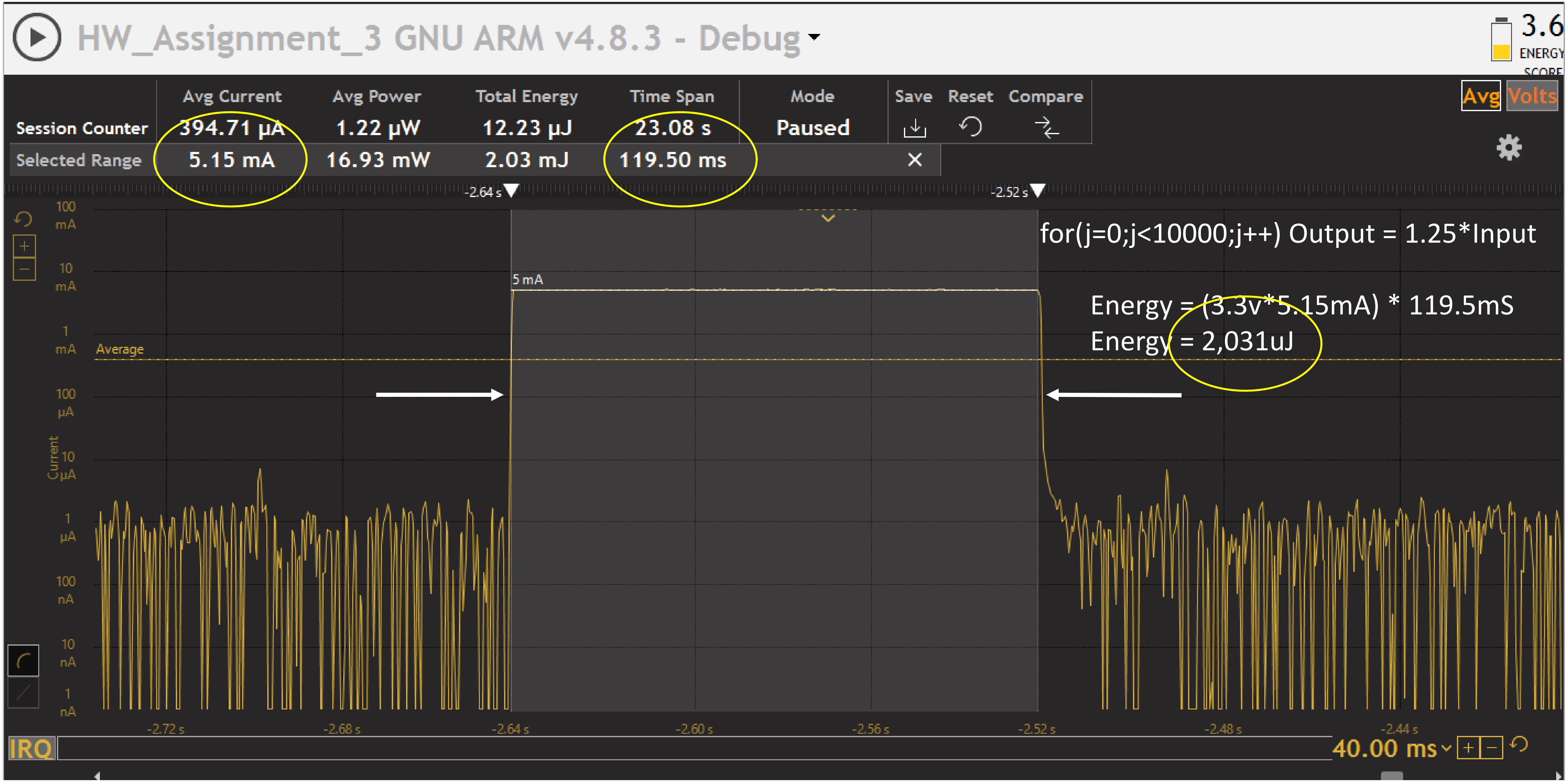
# Cortex-M3: Strength in Integers

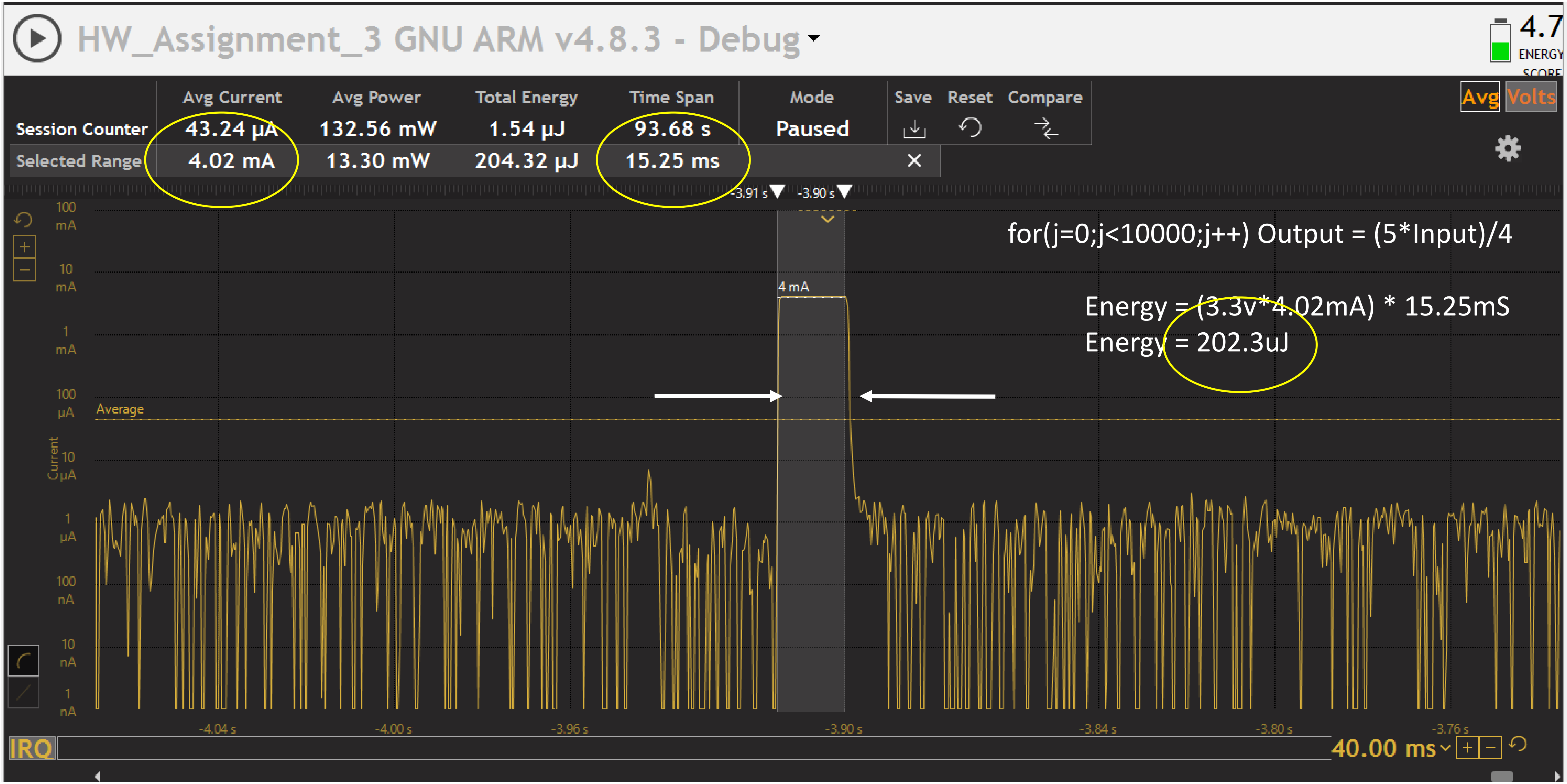
- Expressing a constant as a decimal point versus fractions in Integer math
  - 1.25 versus 5/4
- Is Output = 1.25*Input equivalent to 5/4*Input in Integer math?
  - NO!
    - In 1.25*Input, the rounding occurs after the multiplication (Input = 5, Output = 6)
    - 5/4*Input, the rounding occurs after 5/4 (Input = 5, Output = 5)
  - (5*Input) / 4 is equivalent. (Input = 5, Output = 6)
- Lets determine the relative energy of each method using the following routine:

```
void TestRoutine(int Input){
  int Output;
  int j;

      for(j=0;j<10000;j++) Output = 1.25*Input;

//    for(j=0;j<10000;j++) Output = Input*5/4;
}
```


HW_Assignment_3 GNU ARM v4.8.3 - Debug
3.6
ENERGY SCORE
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Session Counter
394.71 µA
1.22 µW
12.23 µJ
23.08 s
Paused
Selected Range
5.15 mA
16.93 mW
2.03 mJ
119.50 ms
Avg
Volts
-2.64 s
-2.52 s
for(j=0;j<10000;j++) Output = 1.25*Input
5 mA
Energy = (3.3v*5.15mA) * 119.5mS
Energy = 2,031uJ
Average
Current
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
1 nA
-2.72 s
-2.68 s
-2.64 s
-2.60 s
-2.56 s
-2.52 s
-2.48 s
-2.44 s
IRQ
40.00 ms


HW_Assignment_3 GNU ARM v4.8.3 - Debug
4.7
ENERGY SCORE
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Session Counter
43.24 µA
132.56 mW
1.54 µJ
93.68 s
Paused
Selected Range
4.02 mA
13.30 mW
204.32 µJ
15.25 ms
Avg
Volts
-3.91 s
-3.90 s
4 mA
Average
Current
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
1 nA
-4.04 s
-4.00 s
-3.96 s
-3.90 s
-3.84 s
-3.80 s
-3.76 s
IRQ
40.00 ms
for(j=0;j<10000;j++) Output = (5*Input)/4
Energy = (3.3v*4.02mA) * 15.25mS
Energy = 202.3uJ

# Assembly code for Output = 1.25*Input

```
000055a2:   ldr     r0,[sp,#0x4]
000055a4:   bl      0x00005978
000055a8:   mov     r2,r0
000055aa:   mov     r3,r1
000055ac:   mov     r0,r2
000055ae:   mov     r1,r3
000055b0:   mov.w   r2,#0x0
000055b4:   ldr     r3,[pc,#0x30] ; 0x55e4
000055b6:   bl      0x00005a44
000055ba:   mov     r2,r0
000055bc:   mov     r3,r1
000055be:   mov     r0,r2
000055c0:   mov     r1,r3
000055c2:   bl      0x00005e68
000055c6:   mov     r3,r0
000055c8:   str     r3,[sp,#0xc]
00005978:   teq     r0,#0x0
0000597c:   itt     eq
0000597e:   movs    r1,#0x0
00005980:   bx      lr
00005982:   push    {r4,r5,lr}
00005984:   mov.w   r4,#0x400
00005988:   add.w   r4,r4,#0x32
0000598c:   ands    r5,r0,#0x80000000
00005990:   it      mi
00005992:   rsbs    r0,r0,#0
00005994:   mov.w   r1,#0x0
00005998:   b       0x0000000000005818
00005818:   teq     r1,#0x0
0000581c:   itt     eq
0000581e:   mov     r1,r0
00005820:   movs    r0,#0x0
00005822:   clz     r3,r1
00005826:   it      eq
00005828:   adds    r3,#0x20
0000582a:   sub.w   r3,r3,#0xb
0000582e:   subs.w  r2,r3,#0x20
00005832:   bge     0x00000000000584e0
000584e:   it      le
00005850:   rsb.w   r12,r2,#0x20
00005854:   lsl.w   r1,r1,r2
00005858:   lsr.w   r12,r0,r12
0000585c:   itt     le
0000585e:   orr.w   r1,r1,r12
00005862:   lsls    r0,r2
00005864:   subs    r4,r4,r3
00005866:   ittt    ge
00005868:   add.w   r1,r1,r4,lsl #20
0000586c:   orrs    r1,r5
0000586e:   pop     {r4,r5,pc}
00005a44:   push    {r4,r5,r6,lr}
00005a46:   mov.w   r12,#0xff
00005a4a:   orr     r12,r12,#0x700
00005a4e:   ands.w  r4,r12,r1,lsr #20
00005a52:   ittte   ne
00005a54:   ands.w  r5,r12,r3,lsr #20
00005a58:   teq     r4,r12
00005a5c:   teq     r5,r12
00005a60:   bl      0x0000000000005c20
00005a64:   add     r4,r5
00005a66:   eor.w   r6,r1,r3
00005a6a:   bic.w   r1,r1,r12,lsl #21
00005a6e:   bic.w   r3,r3,r12,lsl #21
00005a72:   orrs.w  r5,r0,r1,lsl #12
00005a76:   it      ne
00005a78:   orrs.w  r5,r2,r3,lsl #12
00005a7c:   orr     r1,r1,#0x100000
00005a80:   orr     r3,r3,#0x100000
00005a84:   beq     0x0000000000005af8
00005a86:   umull   r12,lr,r0,r2
00005a8a:   mov.w   r5,#0x0
00005a8e:   umlal   lr,r5,r1,r2
00005a92:   and     r2,r6,#0x80000000
00005a96:   umlal   lr,r5,r0,r3
00005a9a:   mov.w   r6,#0x0
00005a9e:   umlal   r5,r6,r1,r3
00005aa2:   teq     r12,#0x0
00005aa6:   it      ne
00005aa8:   orr     lr,lr,#0x1
00005aac:   sub.w   r4,r4,#0xff
00005ab0:   cmp.w   r6,#0x200
00005ab4:   sbc     r4,r4,#0x300
00005ab8:   bcs     0x0000000000005ac4
00005aba:   lsls.w  lr,lr,#1
00005abe:   adcs    r5,r5
00005ac0:   adc.w   r6,r6,r6
00005ac4:   orr.w   r1,r2,r6,lsl #11
00005ac8:   orr.w   r1,r1,r5,lsr #21
00005acc:   lsl.w   r0,r5,#11
00005ad0:   orr.w   r0,r0,lr,lsr #21
00005ad4:   lsl.w   lr,lr,#11
00005ad8:   subs.w  r12,r4,#0xfd
00005adc:   it      hi
00005ade:   cmp.w   r12,#0x700
00005ae2:   bhi     0x0000000000005b22
00005ae4:   cmp.w   lr,#0x80000000
00005ae8:   it      eq
00005aea:   lsrs.w  lr,r0,#1
00005aee:   adcs    r0,r0,#0x0
00005af2:   adc.w   r1,r1,r4,lsl #20
00005af6:   pop     {r4,r5,r6,pc}
00005e68:   lsl.w   r2,r1,#1
00005e6c:   adds.w  r2,r2,#0x200000
00005e70:   bcs     0x0000000000005e9e
00005e72:   bpl     0x0000000000005e98
00005e74:   mvn     r3,#0x3e0
00005e78:   subs.w  r2,r3,r2,asr #21
00005e7c:   bls     0x0000000000005ea4
00005e7e:   lsl.w   r3,r1,#11
00005e82:   orr     r3,r3,#0x80000000
00005e86:   orr.w   r3,r3,r0,lsr #21
00005e8a:   tst     r1,#0x80000000
00005e8e:   lsr.w   r0,r3,r2
00005e92:   it      ne
00005e94:   rsbs    r0,r0,#0
00005e96:   bx      lr
```



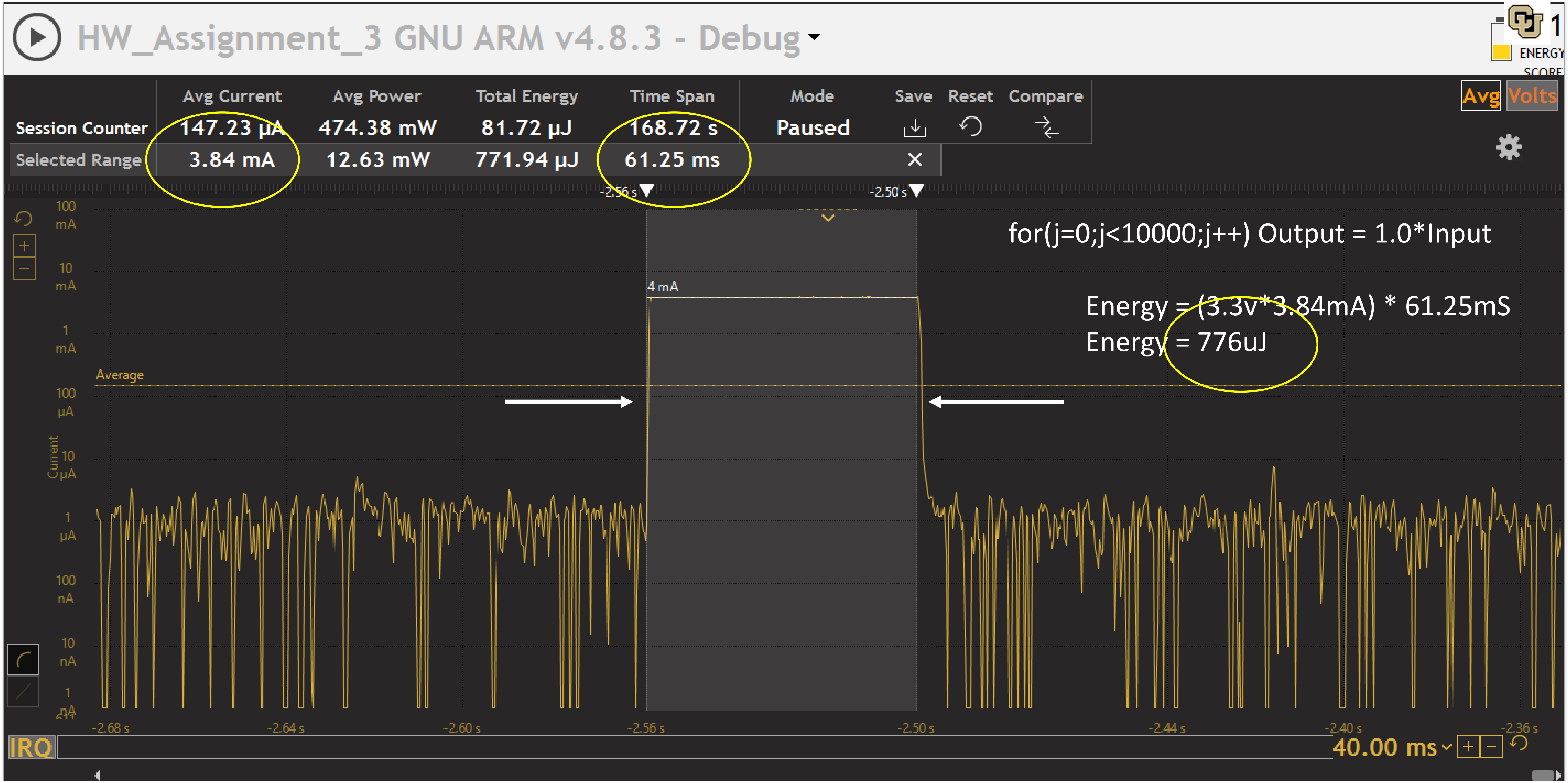
Simplicity Studio disassembler

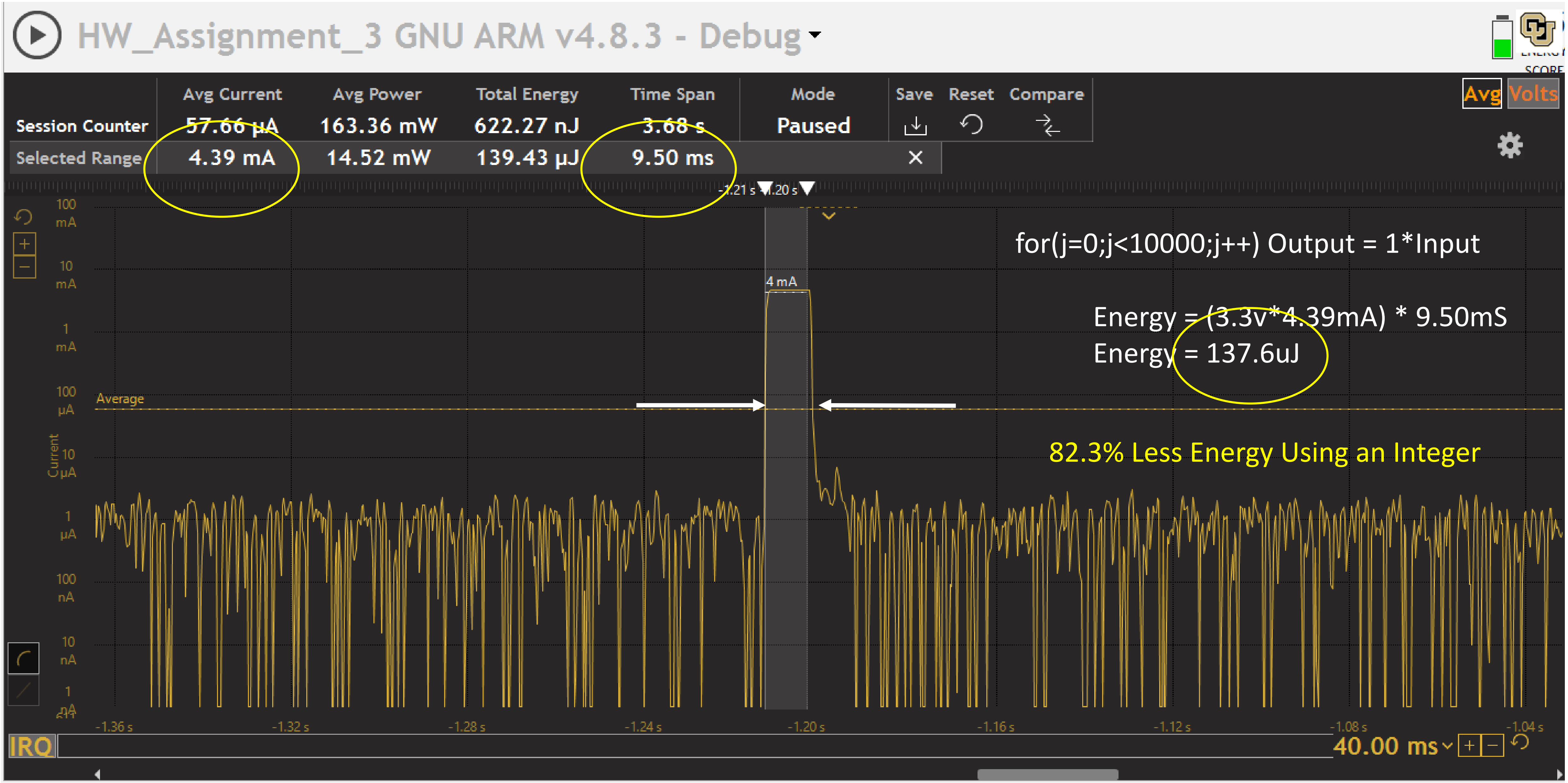
# Assembly code for Output = (5*Input)/4

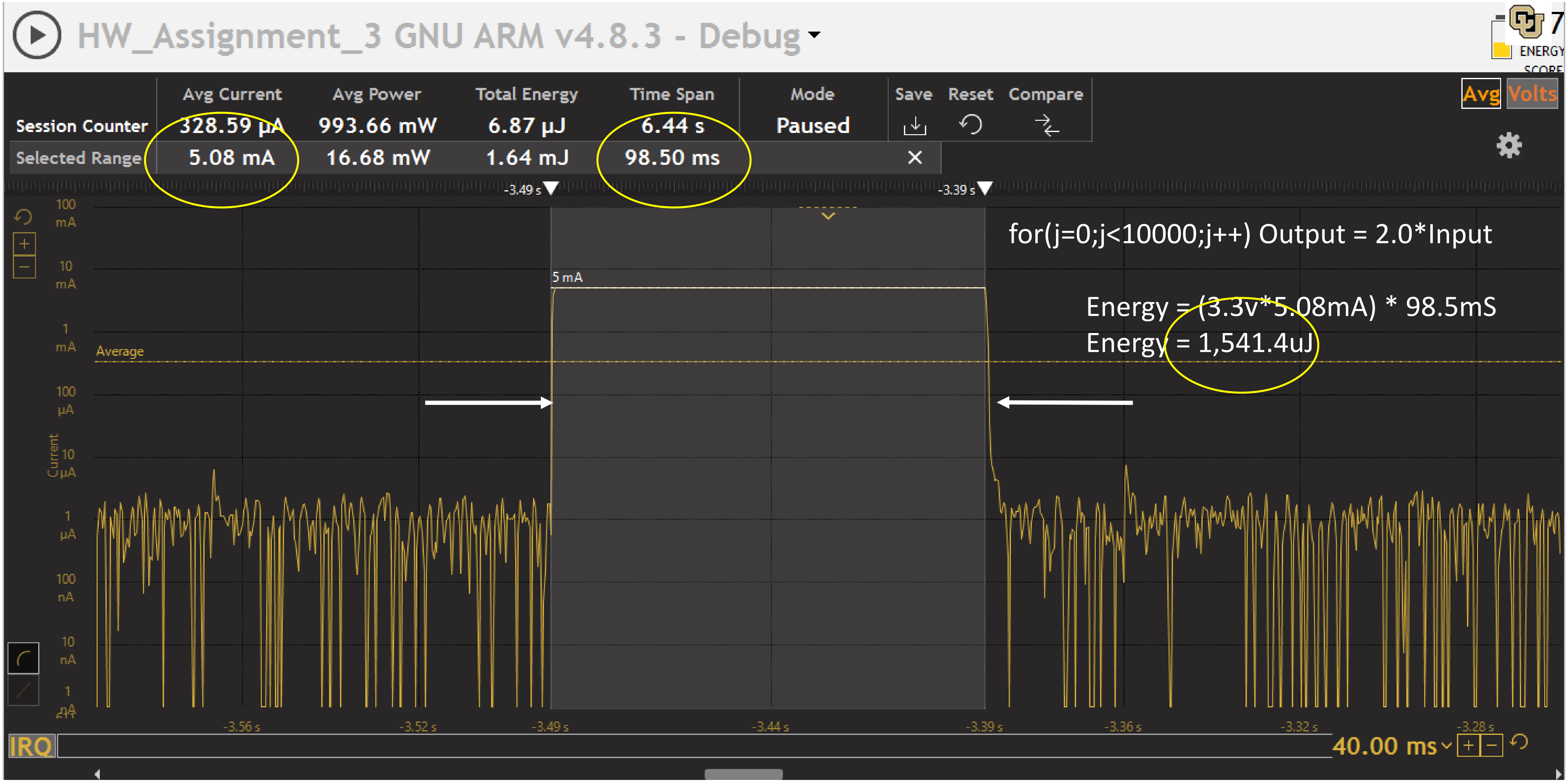
```
000055ca:   ldr     r2,[sp,#0x4]
000055cc:   mov     r3,r2
000055ce:   lsls    r3,r3,#2
000055d0:   add     r3,r2
000055d2:   cmp     r3,#0x0
000055d4:   bge     0x000055d8
000055d6:   adds    r3,#0x3
000055d8:   asrs    r3,r3,#0x2
000055da:   str     r3,[sp,#0xc]
```

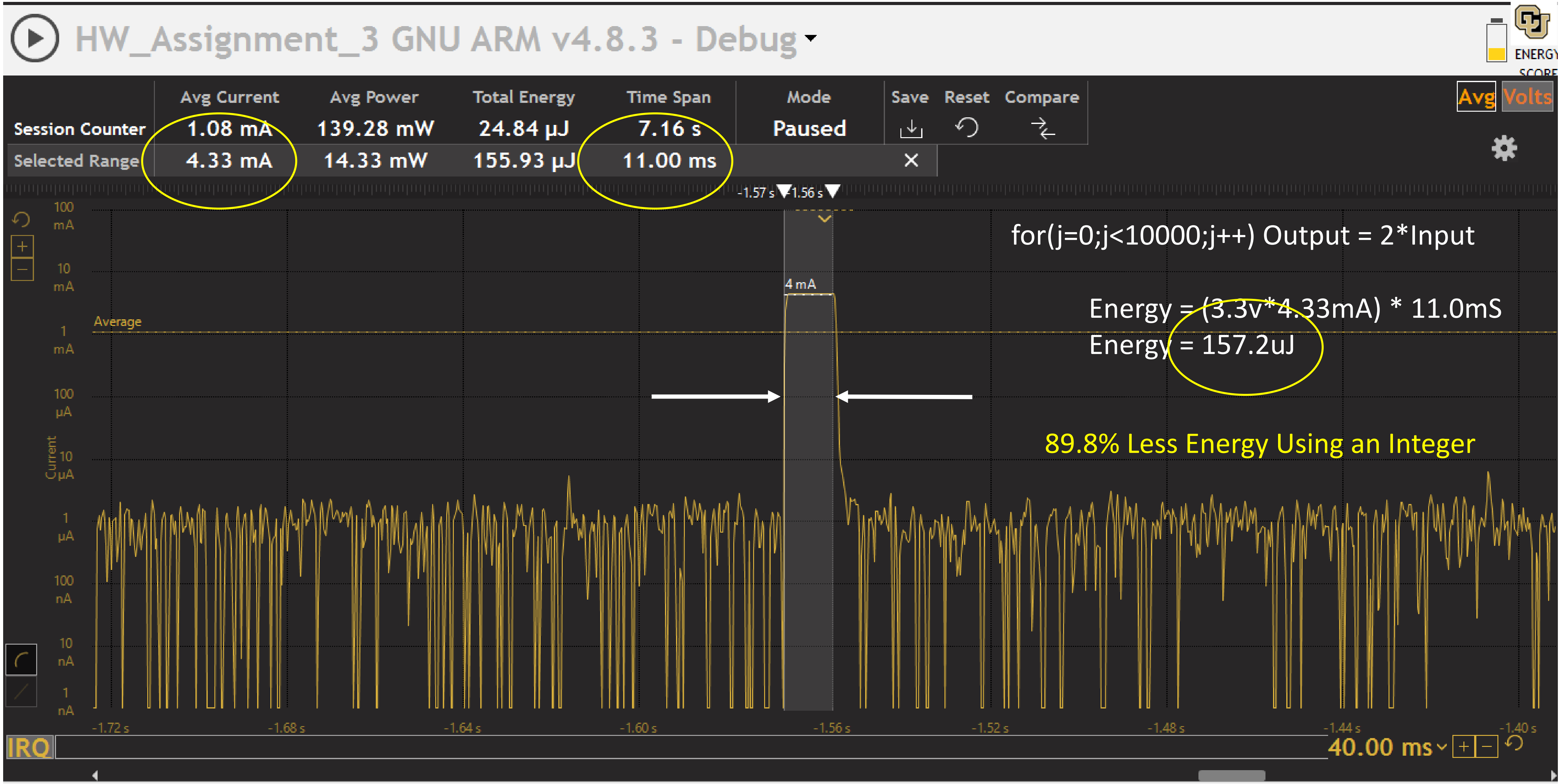
# Cortex-M3: Strength in Integers

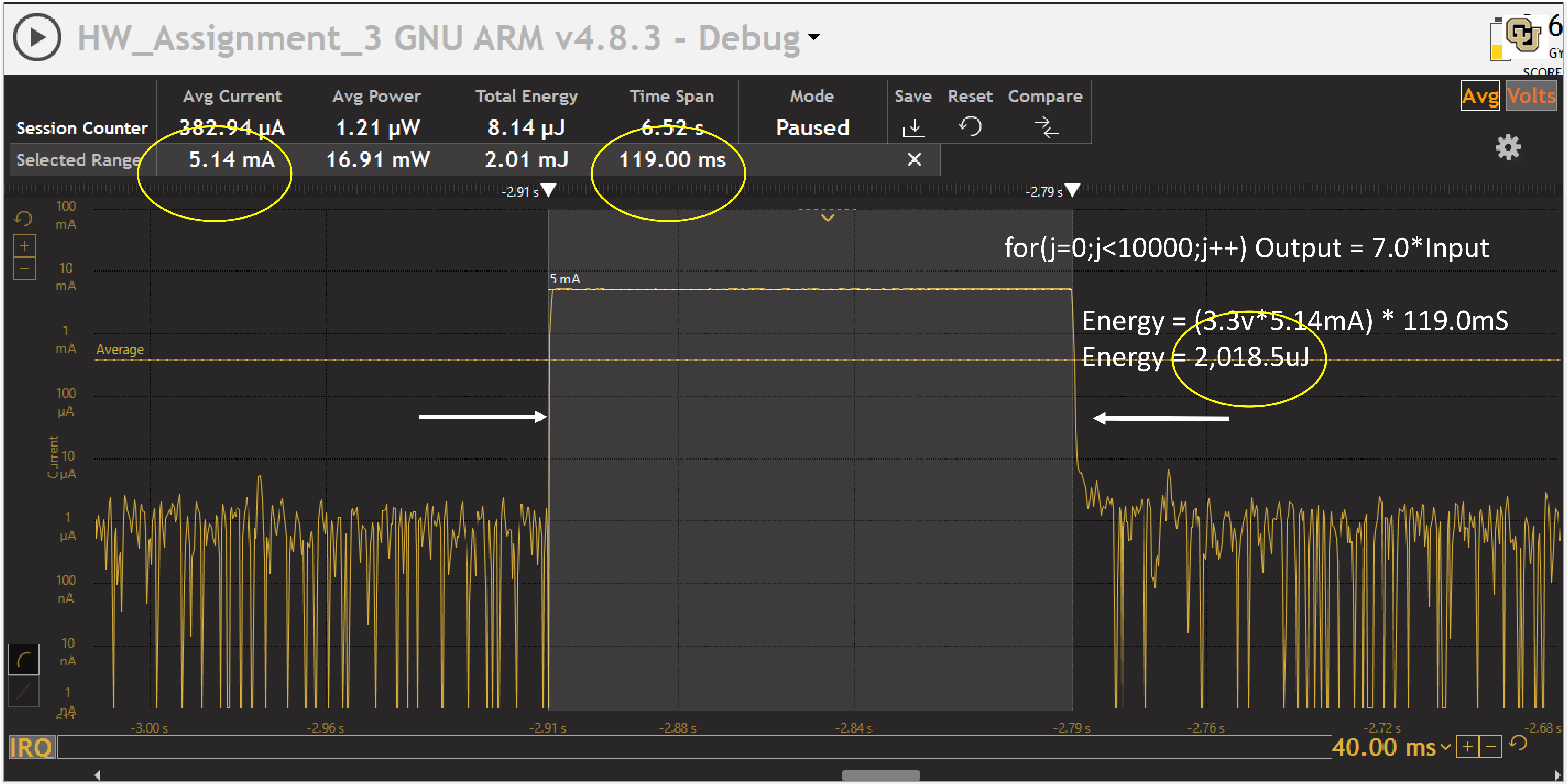
- Converting decimal points to whole number fractions where possible results in significant Energy Savings!
  - Energy for 1.25*Input = 2031.0uJ
  - Energy for (5*Input) / 4 = 202.3uJ
  - A 90% savings in Energy by going to Whole Fractions!
- A more common mistake or error may be specifying a whole number constant as a decimal or floating point number
  - Lets experiment with 1/1.0, 2/2.0, and 7/7.0

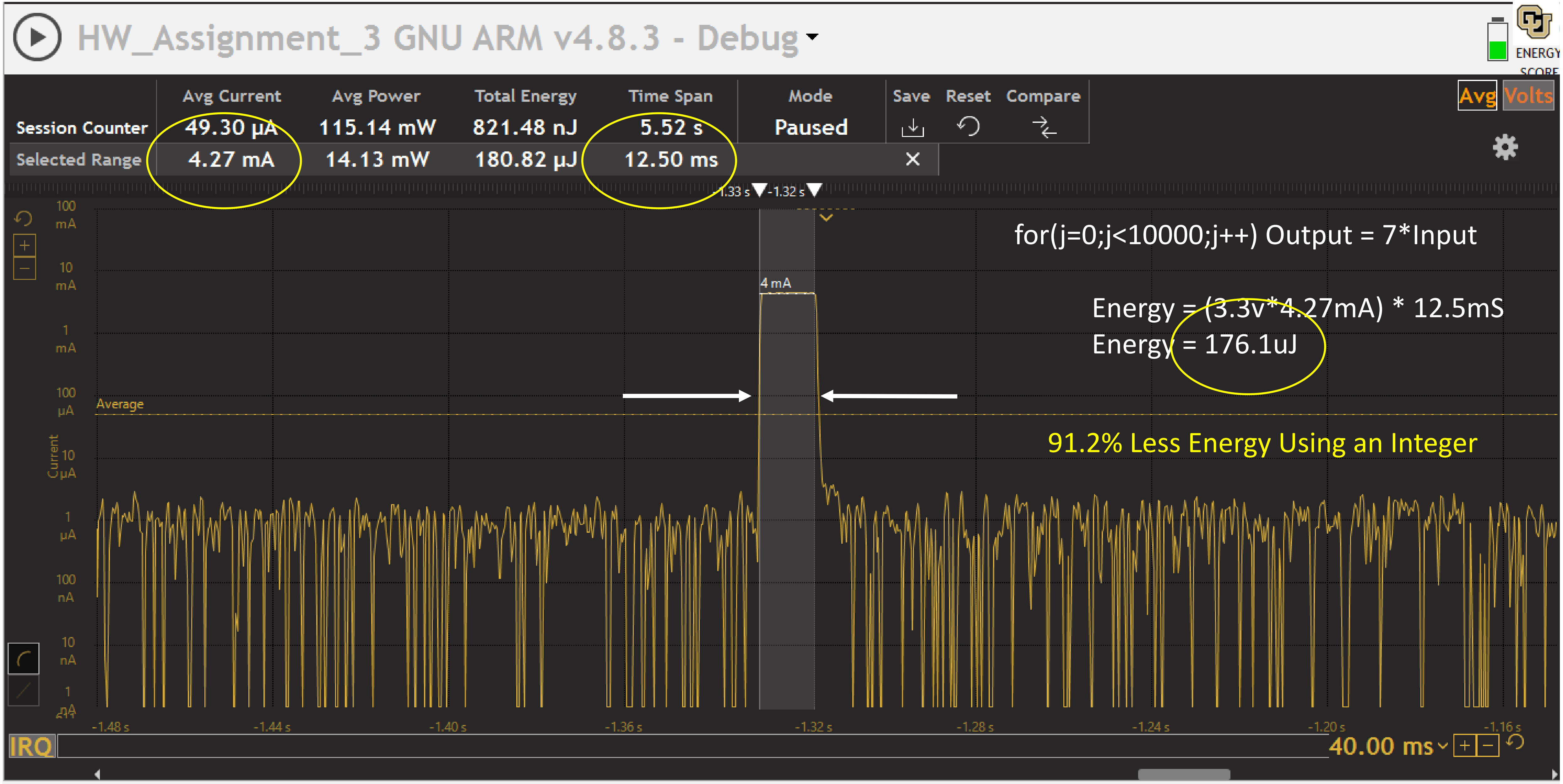

HW_Assignment_3 GNU ARM v4.8.3 - Debug
ENERGY SCORE
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Avg
Volts
Session Counter 147.23 µA 474.38 mW 81.72 µJ 168.72 s Paused
Selected Range 3.84 mA 12.63 mW 771.94 µJ 61.25 ms
-2.56 s
-2.50 s
for(j=0;j<10000;j++) Output = 1.0*Input
Energy = (3.3v*3.84mA) * 61.25mS
Energy = 776uJ
4 mA
Average
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
1 nA
Current
-2.68 s
-2.64 s
-2.60 s
-2.56 s
-2.50 s
-2.44 s
-2.40 s
-2.36 s
IRQ
40.00 ms


HW_Assignment_3 GNU ARM v4.8.3 - Debug
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Avg
Volts
Session Counter
57.66 µA
163.36 mW
622.27 nJ
3.68 s
Paused
Selected Range
4.39 mA
14.52 mW
139.43 µJ
9.50 ms
-1.21 s
-1.20 s
4 mA
Average
Current
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
1 nA
-1.36 s
-1.32 s
-1.28 s
-1.24 s
-1.20 s
-1.16 s
-1.12 s
-1.08 s
-1.04 s
IRQ
40.00 ms
for(j=0;j<10000;j++) Output = 1*Input
Energy = (3.3v*4.39mA) * 9.50mS
Energy = 137.6uJ
82.3% Less Energy Using an Integer


HW_Assignment_3 GNU ARM v4.8.3 - Debug
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Session Counter
328.59 µA
993.66 mW
6.87 µJ
6.44 s
Paused
Selected Range
5.08 mA
16.68 mW
1.64 mJ
98.50 ms
Avg
Volts
5 mA
Average
IRQ
40.00 ms
for(j=0;j<10000;j++) Output = 2.0*Input
Energy = (3.3v*5.08mA) * 98.5mS
Energy = 1,541.4uJ

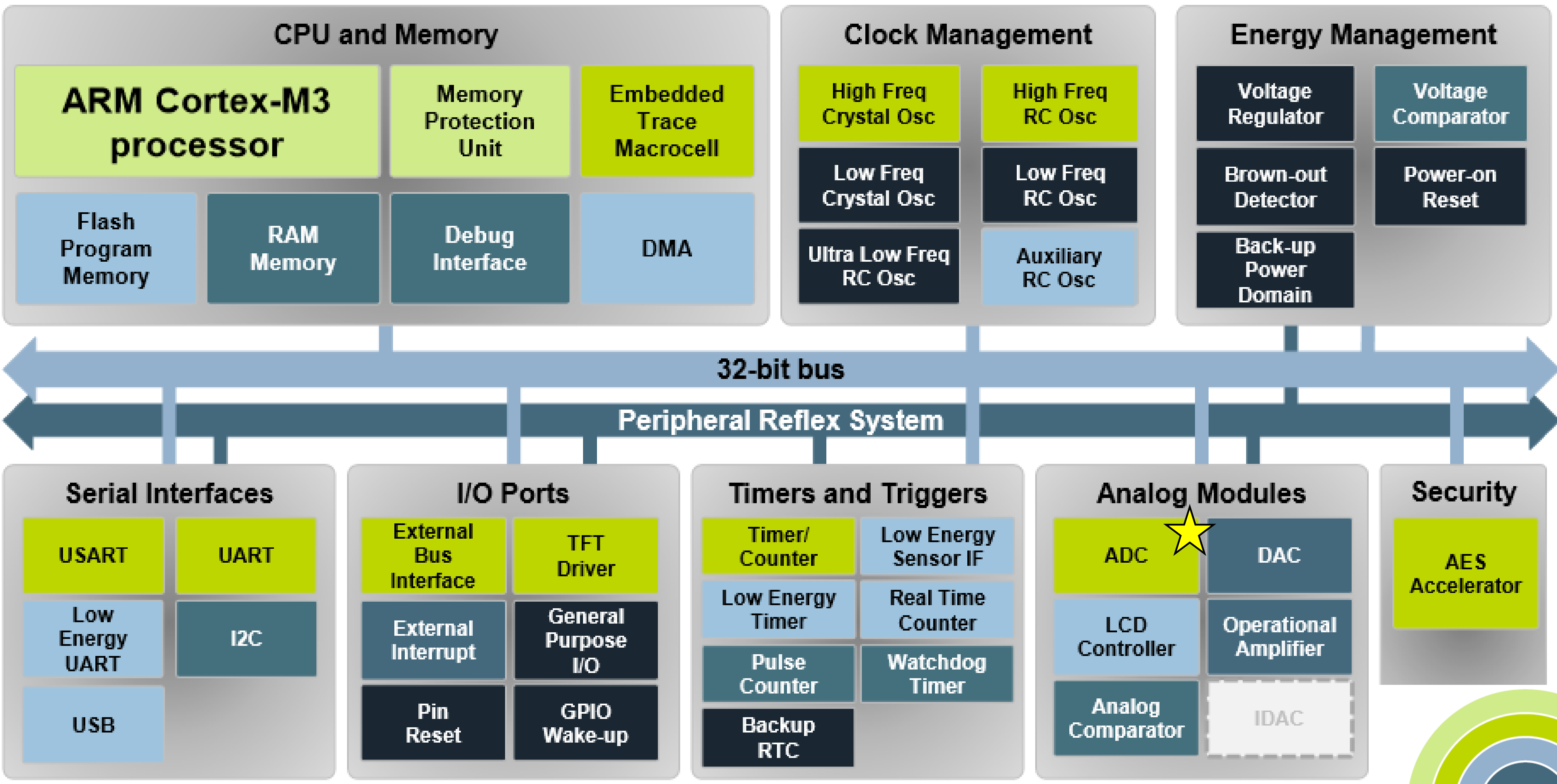

HW_Assignment_3 GNU ARM v4.8.3 - Debug
ENERGY SCORE
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Session Counter
1.08 mA
139.28 mW
24.84 µJ
7.16 s
Paused
Selected Range
4.33 mA
14.33 mW
155.93 µJ
11.00 ms
Avg
Volts
-1.57 s
-1.56 s
4 mA
Average
Current
IRQ
40.00 ms
for(j=0;j<10000;j++) Output = 2*Input
Energy = (3.3v*4.33mA) * 11.0mS
Energy = 157.2uJ
89.8% Less Energy Using an Integer

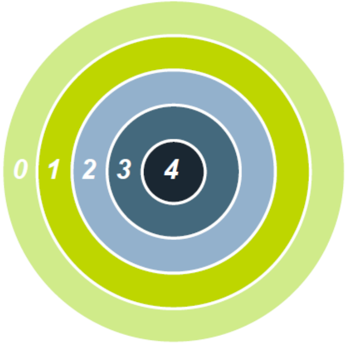

HW_Assignment_3 GNU ARM v4.8.3 - Debug
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Session Counter
382.94 µA
1.21 µW
8.14 µJ
6.52 s
Paused
Selected Range
5.14 mA
16.91 mW
2.01 mJ
119.00 ms
Avg
Volts
-2.91 s
-2.79 s
5 mA
Average
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
1 nA
Current
-3.00 s
-2.96 s
-2.91 s
-2.88 s
-2.84 s
-2.79 s
-2.76 s
-2.72 s
-2.68 s
IRQ
40.00 ms
for(j=0;j<10000;j++) Output = 7.0*Input
Energy = (3.3v*5.14mA) * 119.0mS
Energy = 2,018.5uJ


HW_Assignment_3 GNU ARM v4.8.3 - Debug
ENERGY SCORE
Avg Current
Avg Power
Total Energy
Time Span
Mode
Save
Reset
Compare
Avg
Volts
Session Counter
49.30 µA
115.14 mW
821.48 nJ
5.52 s
Paused
Selected Range
4.27 mA
14.13 mW
180.82 µJ
12.50 ms
-1.33 s
-1.32 s
4 mA
Average
Current
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
1 nA
-1.48 s
-1.44 s
-1.40 s
-1.36 s
-1.32 s
-1.28 s
-1.24 s
-1.20 s
-1.16 s
IRQ
40.00 ms
for(j=0;j<10000;j++) Output = 7*Input
Energy = (3.3v*4.27mA) * 12.5mS
Energy = 176.1uJ
91.2% Less Energy Using an Integer

# Cortex-M3: Strength in Integers

- In summary, the Cortex-M3 is much more efficient using integers
  - Where possible, express decimals as fractions
    - To be equivalent, the division should be the last operation
  - Make sure integer constants are not mistakenly become floating point numbers by adding a decimal point!
  - 82 to 91% less energy used or 5.6 to 11.5x more Energy efficient!
- Insure that the constants and variables types match the end results



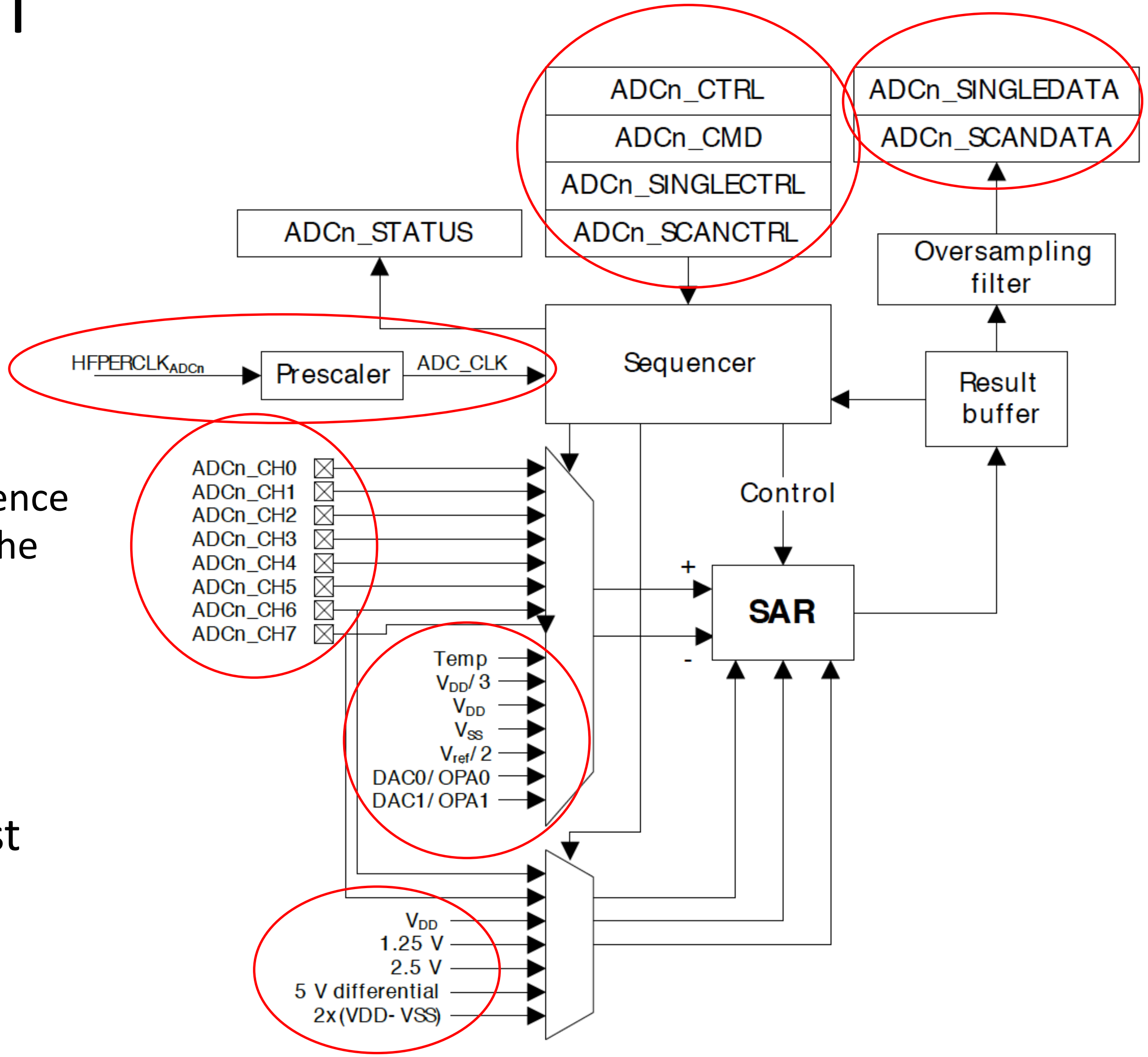






Silicon Labs' EFM32LG Reference Manual

# ADC – Analog Digital Converter

- 13 MHz to 32 kHz allowed for ADC_CLK
- Maximum 1 MSPS @ 12-bit
- Maximum 1.86 MSPS @ 6-bit
- Programmable scan sequence
  - Up to 8 configurable samples in scan sequence
  - Mask to select which pins are included in the sequence
  - Triggered by software or PRS input
  - One shot or repetitive mode
  - Oversampling available
  - Overflow interrupt flag set
- Interrupt generation and/or DMA request
  - Finished single conversion
  - Finished scan conversion
  - Single conversion results overflow
  - Scan sequence results overflow



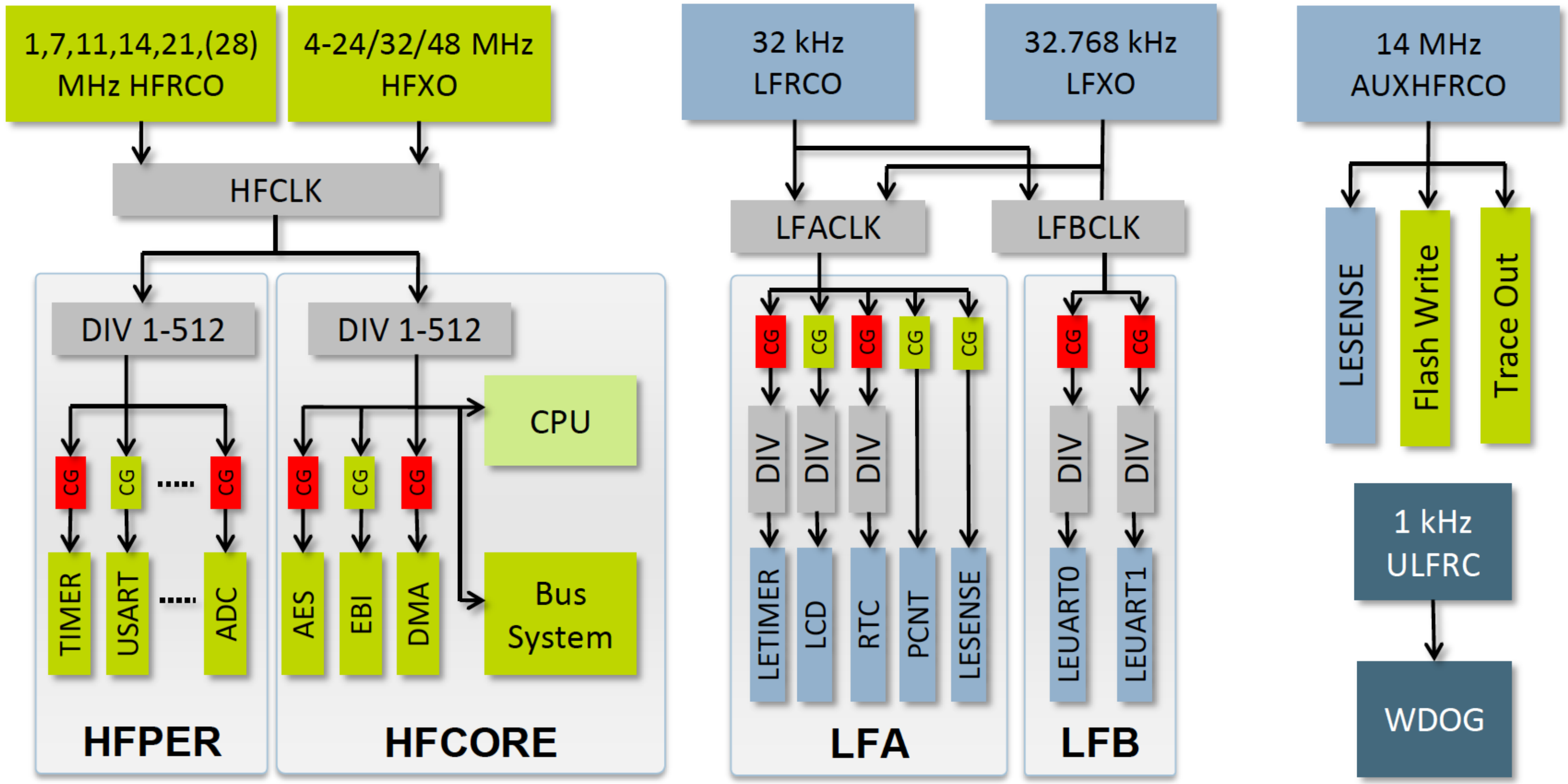



Silicon Labs' EFM32LG data sheet

# Clocks and Oscillators

ADC is connected to the HFPERCLK, which is always running in EM0 & EM1, so what does this mean?

No need to enable an oscillator or to tie the oscillator to a clock branch




But, you still need to enable the clock to the ADC & set the ADC Prescalar!

# Setting up the ADC

- First, the clock tree to the ADC must be established
  - Without establishing the clock tree, all writes to the ADCn registers will not occur
  - ADCn clock source is the HFPERCLK, so no oscillator enable is required
  - Pseudo code in the CMU setup routine to enable the ADCn clock tree:
    - Lastly, enable the ADC clocking using the CMU_ClockEnable for the ADCn
  - ADC maximum clock frequency is 13MHz
    - With the standard HFRCO set at 14MHz, the ADC Prescalar must be greater than 1!
    - The ADC has its own Prescalar from 1 to 128.  These are NOT to the power of 2.
    - This will be set up in the ADC_Init() call

# ADC - Conversions

- Conversion is comprised of 2 phases
  - Input is sampled during the acquisition phase
    - The acquisition times can be set to any integer power of 2 from 1 to 256 ADC_CLK cycles
  - Converted to digital representation during the approximation phase
- The analog to digital converter core uses one clock cycle per output bit in the approximation phase.

***ADC Total Conversion Time (in ADC_CLK cycles) Per Output***

$$T_{conv} = (T_A + N) \times OSR$$

- $T_A$ = Acquisition Time, N = number of bits, OSR = Oversampling ratio

# Calculating MSPS

***ADC Total Conversion Time (in ADC_CLK cycles) Per Output***

$$T_{conv} = (T_A + N) \times OSR$$

- HFRCO = 14 MHz
- Prescalar = 4
- 12 bit conversion
- Acquisition time > 3uS
- No Oversampling => OSR = 1
- ADC_clock = 14MHz/4 = 3.5MHz
- Acquisition time
  - $3 \times 10^{-6} / (1/3.5 \times 10^6)$ = 10.5 clock cycles
  - $T_A$ = 11 clock cycles
- 12-bit conversion
  - N = 12

- $T_{conv} = (T_A + N) \times OSR$
- $T_{conv} = (11+12) \times 1$
- $T_{conv}$ = 23 ADC_Clocks
- $T_{conv} = 23 \times (1/3.5 \times 10^6)$
- $T_{conv}$ = 6.57uS
- MSPS = 1/ $T_{conv}$
- MSPS = 1/6.57uS
- MSPS = 0.152 or 152 KSPS

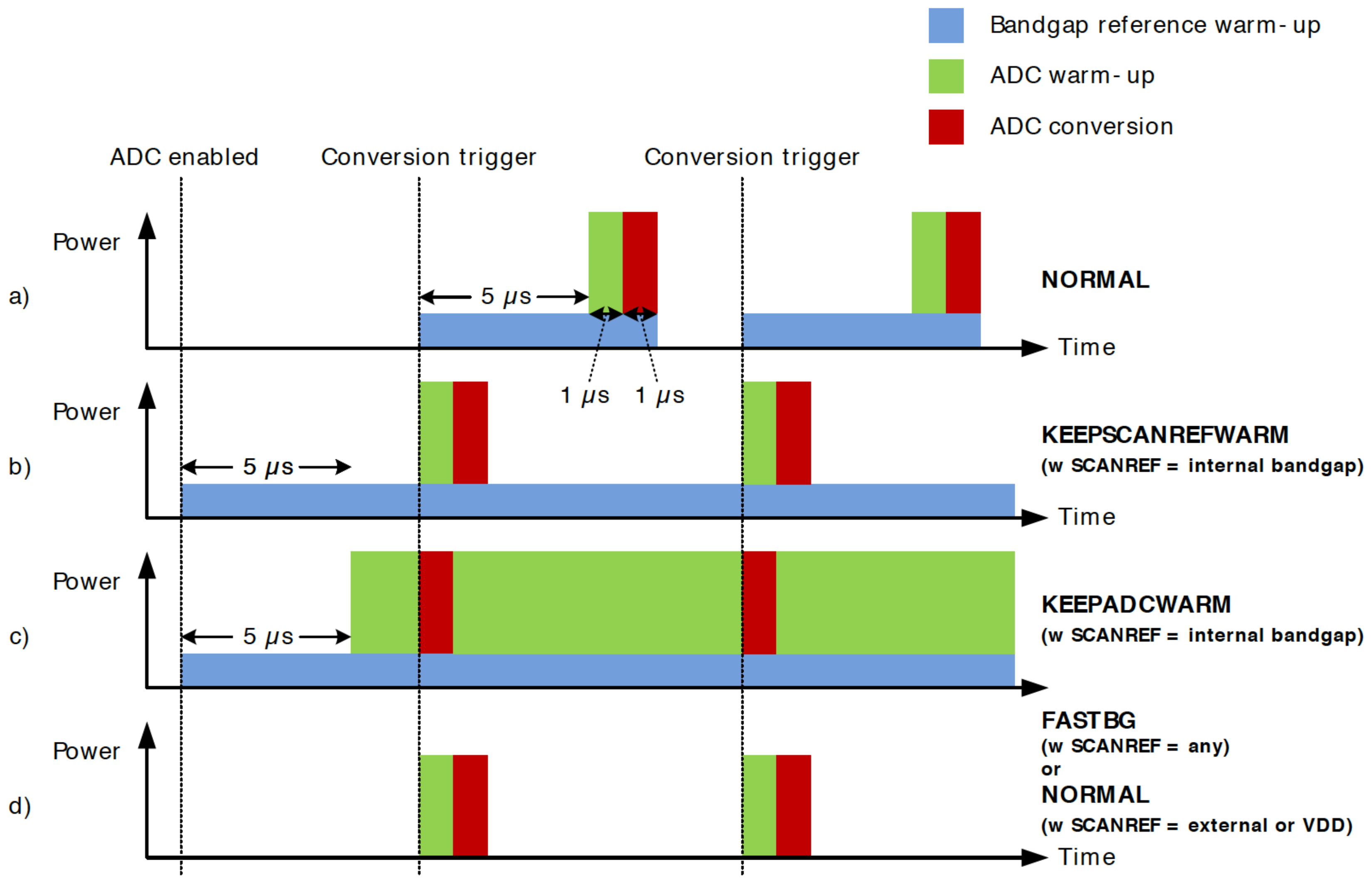
# ADC – Warm-up Time

- The ADC needs to be warmed up some time before a conversion can take place. This time period is called the warm-up time
- When enabling the ADC or changing references between samples, the ADC is automatically warmed up for 1µs and an additional 5 µs if the bandgap is selected as reference
- Normally, the ADC will shutdown to conserve energy between conversions. To reduce latency, the ADC can be kept warmed between conversions by setting the warm-up mode in the ADCn_CTRL register.
  - NORMAL: ADC and references are shut off when no samples are waiting
  - FASTBG: Bandgap warm-up is eliminated at the expense of reference accuracy
  - KEEPSCANREFWARM: The reference selected for scan mode is kept warm
  - KEEPADCWARM: The ADC and reference selected for scan mode is kept warm

# ADC – Warm-up Time

- The warm-up timing is done automatically by the ADC, given that a proper time base is given in the TIMEBASE bits in ADCn_CTRL.
- The TIMEBASE must be set to the number of HFPERCLK which corresponds to at least 1 µs.
  - To set the Time base, the following library routine can be used:
    - ADC_TimebaseCalc(0);
- When entering Energy Modes 2 or 3, the ADC must be stopped and WARMUPMODE in ADCn_CTRL written to 0.

## *Figure 28.3. ADC Analog Power Consumption With Different WARMUPMODE Settings*



# ADC – Temperature Measurement

- The ADC includes an internal Temperature Sensor which is characterized during production.
- The production characterization data can be found in the Device Information page which includes the readout from the ADC at production temperature, ADC0_TEMP_0_READ_1V25. The production temperature, CAL_TEMP_0 is also provided on the Device Information Page.
- To determine the temperature, take a measurement of the temperature input using the 1.25v reference in 12 bit mode. The equation to convert the 12-bit result to degrees C is:

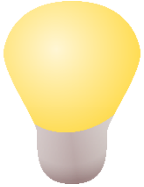
***ADC Temperature Measurement***

$$T_{CELSIUS}=CAL\_TEMP\_0-(ADC0\_TEMP\_0\_READ\_1V25-ADC\_result)\times Vref/(4096\times TGRAD\_ADCTH) \tag{28.2}$$

- TGRAD_ADCTH can be found in the Leopard Gecko datasheet

# ADC – Temperature Measurement

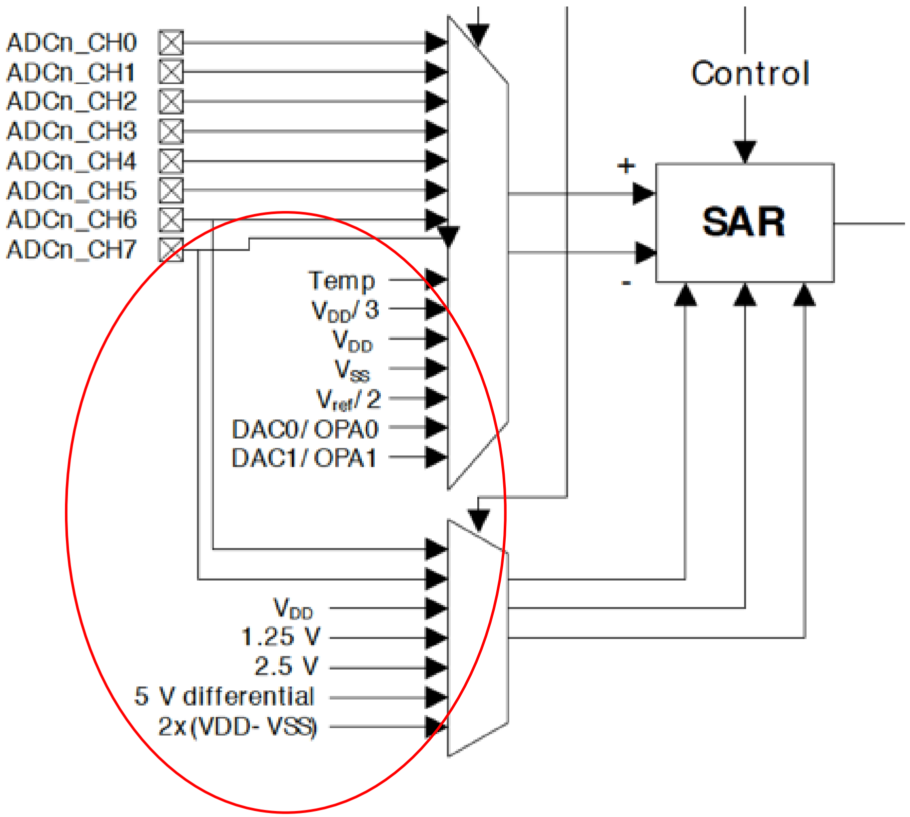
```
float convertToCelsius(int32_t adcSample)
{
  float temp;
  /* Factory calibration temperature from device information page. */
  float cal_temp_0 = (float)((DEVINFO->CAL & _DEVINFO_CAL_TEMP_MASK)
                             >> _DEVINFO_CAL_TEMP_SHIFT);
  float cal_value_0 = (float)((DEVINFO->ADC0CAL2
                               & _DEVINFO_ADC0CAL2_TEMP1V25_MASK)
                              >> _DEVINFO_ADC0CAL2_TEMP1V25_SHIFT);
  /* Temperature gradient (from datasheet) */
  float t_grad = -6.27;
  temp = (cal_temp_0 - ((cal_value_0 - adcSample)  / t_grad));
  return temp;
}
```



Must give credit in your code to Silicon Labs for this IP by providing comment similar to the sleep routines

# ADC – Reference Selection

- The reference voltage can be selected from these sources:
  - 1.25 V internal bandgap.
  - 2.5 V internal bandgap.
  - VDD.
  - 5 V internal differential bandgap.
  - External single ended input from Ch. 6.
  - Differential input, 2x(Ch. 6 - Ch. 7).
  - Unbuffered 2xVDD.
  - The 2.5 V reference needs a supply voltage higher than 2.5 V.
  - The differential 5 V reference needs a supply voltage higher than 2.75 V.



# ADC - Modes

- Two separate programmable modes, single sample and scan mode. If a single sample conversion is requested while scan mode is active, the single sample conversion will be interleaved between two scan mode conversions
  - Single Sample Mode: Converts a single sample (single input) once per trigger or repetitively. The result can be read out of the ADCn_SINGLEDATA register.
  - Scan Mode: Sweeps through a predefined sequence of the inputs set in the ADCn_SCANCTRL register. The result of the conversions can be found in the ADCn_SCANDATA register.
  - Conversion Tailgating: By setting the TAILGATE bit in the ADCn_CTRL register, Single Sample scans will not begin until the Scan Mode scan has completed. This will minimize the noise in the system for more accurate conversions.

# ADC – Conversion Triggers

- ADC conversions can be triggered by:
  - Writing setting the SINGLESTART or SCANSTART in the ADCn_CMD register
  - Peripheral Reflex System, PRS, inputs can also be used to trigger the start of a scan
  - To enable repetitive scans, the REP bit must be set in the ADCn_SINGLECTRL and ADCn_SCANCTRL registers
  - A scan can be stopped by setting the SINGLESTOP or SCANSTOP bits in the ADCn_CMD register

# ADC - Oversampling

- Oversampling is a method to increase the resolution of an Analog Digital Converter. By taking multiple readings of the input, the effective resolution of the ADC can be enhanced.
- The Leopard Gecko enables Oversampling from 2 to 4096.
- Effectively increasing the resolution of the 12-bit ADC peripheral to 16 bits!

*Table 28.3. Oversampling Result Shifting and Resolution*

| Oversampling setting | # right shifts | Result Resolution # bits |
|---|---|---|
| 2x | 0 | 13 |
| 4x | 0 | 14 |
| 8x | 0 | 15 |
| 16x | 0 | 16 |
| 32x | 1 | 16 |
| 64x | 2 | 16 |
| 128x | 3 | 16 |
| 256x | 4 | 16 |
| 512x | 5 | 16 |
| 1024x | 6 | 16 |
| 2048x | 7 | 16 |
| 4096x | 8 | 16 |

# ADC – Interrupts & DMA

- Separate interrupts for Single and Scan Modes
  - Single Sampling Mode:
    - SINGLE:  Single conversion complete
    - SINGLEOF:  Single result overflow
  - Scan Mode:
    - SCAN: Scan conversion complete
    - SCANOF:  Scan result overflow
- The ADC has two DMA request lines, SINGLE and SCAN, which are set when a single or scan conversion has completed.
  - The request are cleared when the corresponding single or scan result register is read.

# Setting up the ADC

- Second, the ADCn must be set up
  - Inputs to the ADCn must be specified
    - Specify the Input source to the Analog to Digital Converter
      - If external to the MCU, the appropriate GPIO pin must be configured
      - ADCn external GPIO pins should be set to gpioModeDisabled
  - Initialize the ADCn basic parameters
    - ADCn clock prescale value (remember, it is NOT to the power of 2)
    - The over sample rate
    - Time base for the ADC
    - Warmup
    - ADC_Init();

# Setting up the ADC

- Second, the ADCn must be set up (continue)
  - Set up ADC conversion
    - Single conversion or continuous scan
    - Specify the Input source to the Analog to Digital Converter
    - Specify the Reference source to the Analog to Digital Converter
      - Match the reference properly to the voltage range of the input
    - Define the conversion resolution
    - Single ended or differential
    - Repetitive scan
    - ADC conversion resolution
    - Single scan
      - ADC_InitSingle();
    - Scan sequence
      - ADC_InitScan();

# Setting up the ADC

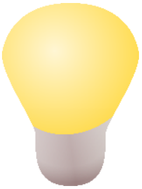
- Third, the DMA channel must be configured / initialized if DMA is being used with the ADC
  - Configure the DMA channel
    - DMA priority
    - DMA request source
    - DMA call back routine
    - DMA_CfgChannel();
  - Configure the DMA descriptors
    - Size of DMA transfers
    - DMA arbitration setting
    - Incrementing of source or destination addresses
    - DMA_CfgDescr();

More on this on Thursday's lecture

More on this on Thursday's lecture

# Setting up the ADC

- Forth, the ADCn interrupts must be enabled if needed
  - Clear all interrupts from the ADCn to remove any interrupts that may have been set up inadvertently by accessing the ADCn->IFC register or the emlib routine
  - Enable the desired interrupts by setting the appropriate bits in ADCn->IEN
  - Set BlockSleep mode to the desired Energy Mode
    - ADCn can be set to operate down to EM1
  - Enable interrupts to the CPU by enabling the ADCn in the Nested Vector Interrupt Control register using NVIC_EnableIRQ(ADCn_IRQn);



# Setting up the ADC

- Fifth, the ADCn interrupt handler must be included
  - Routine name must match the vector table name:

```
Void ADCn_IRQHandler(void) {
}
```

  - Inside this routine, you add the functionality that is desired for the ADCn interrupts