ECEN 5823-001 / -001B

Internet of Things Embedded Firmware

Lecture #6

14 September 2017

Be Boulder.

University of Colorado Boulder

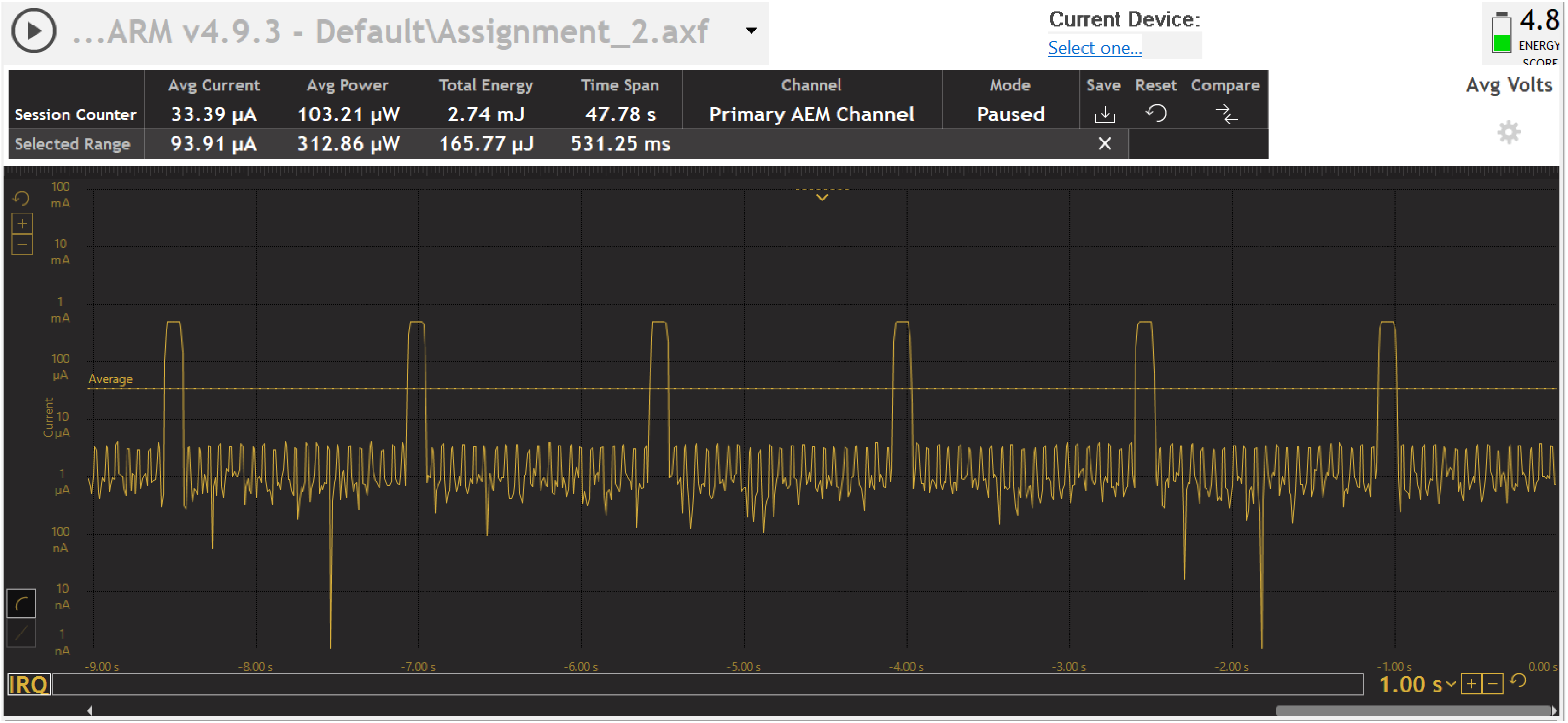
Agenda

- Class announcements
- Review of Managing Energy Mode Managing Energy Modes Assignment
- Assigning Low Energy ADC Assignment
- Finishing summarizing Low Power RF Networks
- ADCs
- Integer Math 101

Class Announcements

- Quiz #3 is due at 11:59pm on Sunday, September 17th, 2017
- Low Energy ADC Assignment is due at 11:59pm on Wednesday, September 20th, 2017
- NOTE: In Simplicity, there are only a limited number of hardware breakpoints. If you run out, you must delete a previous break point to place the new one.

Managing Energy Mode Review



...ARM v4.9.3 - Default\Assignment_2.axf

Current Device: Select one...

4.8 ENERGY SCORE

Avg Volts

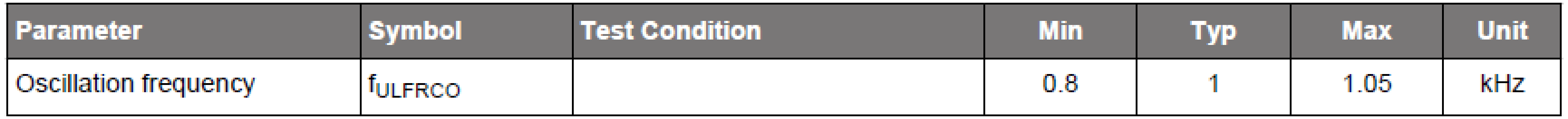
	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode	Save	Reset	Compare
Session Counter	33.39 µA	103.21 µW	2.74 mJ	47.78 s	Primary AEM Channel	Paused			
Selected Range	93.91 µA	312.86 µW	165.77 µJ	531.25 ms					

Managing Energy Mode Review

- Question 1: EM0
  - Energy score: 2.5-2.7
  - Current LED off: ~2.3 - ~2.9mA
  - Current LED on: current in (ii) plus 0.45 to 0.55mA
- Question 2: EM1
  - Energy score: 2.7-2.9
  - Current LED off: ~1.6 - ~2.1mA
  - Current LED on: current in 2(ii) plus 0.45 to 0.55mA
- Question 3: EM2
  - Energy score: 5.2-5.4
  - Current LED off: ~1.5 - ~2.1uA
  - Current LED on: current in 3(ii) plus 0.45 to 0.55mA
- Question 4: EM2: Period and On-time
  - Period: 2.48 – 2.52 S
  - On-time: 49.75 – 50.25 mS
- Question 5: EM3
  - Energy score: 5.2-5.4
  - Current LED off: should be ~1 to 1.4uA less than 3(ii)
  - Current LED on: current in 5(ii) plus 0.45 to 0.55mA
- Question 6: EM3: Period and On-time

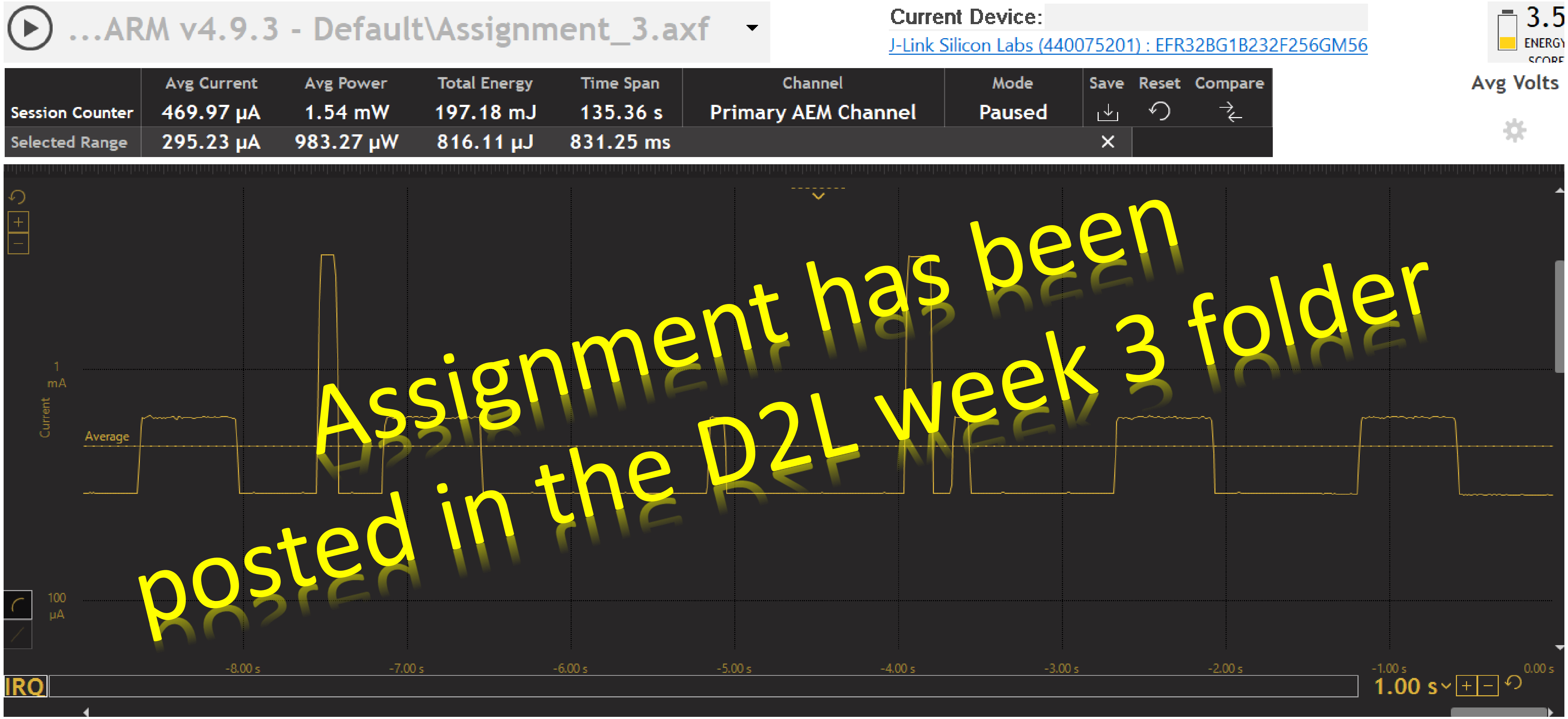
  2.48 – 2.52 S

  49.75 – 50.25 mS



Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Oscillation frequency	f_{ULFRCO}		0.8	1	1.05	kHz

Low Energy ADC Assignment



…ARM v4.9.3 - Default\Assignment_3.axf

Current Device: J-Link Silicon Labs (440075201) : EFR32BG1B232F256GM56

	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode	Save	Reset	Compare
Session Counter	469.97 μA	1.54 mW	197.18 mJ	135.36 s	Primary AEM Channel	Paused			
Selected Range	295.23 μA	983.27 μW	816.11 μJ	831.25 ms					

Assignment has been posted in the D2L week 3 folder

STK6101C – BGM121 Blue Gecko dev kit

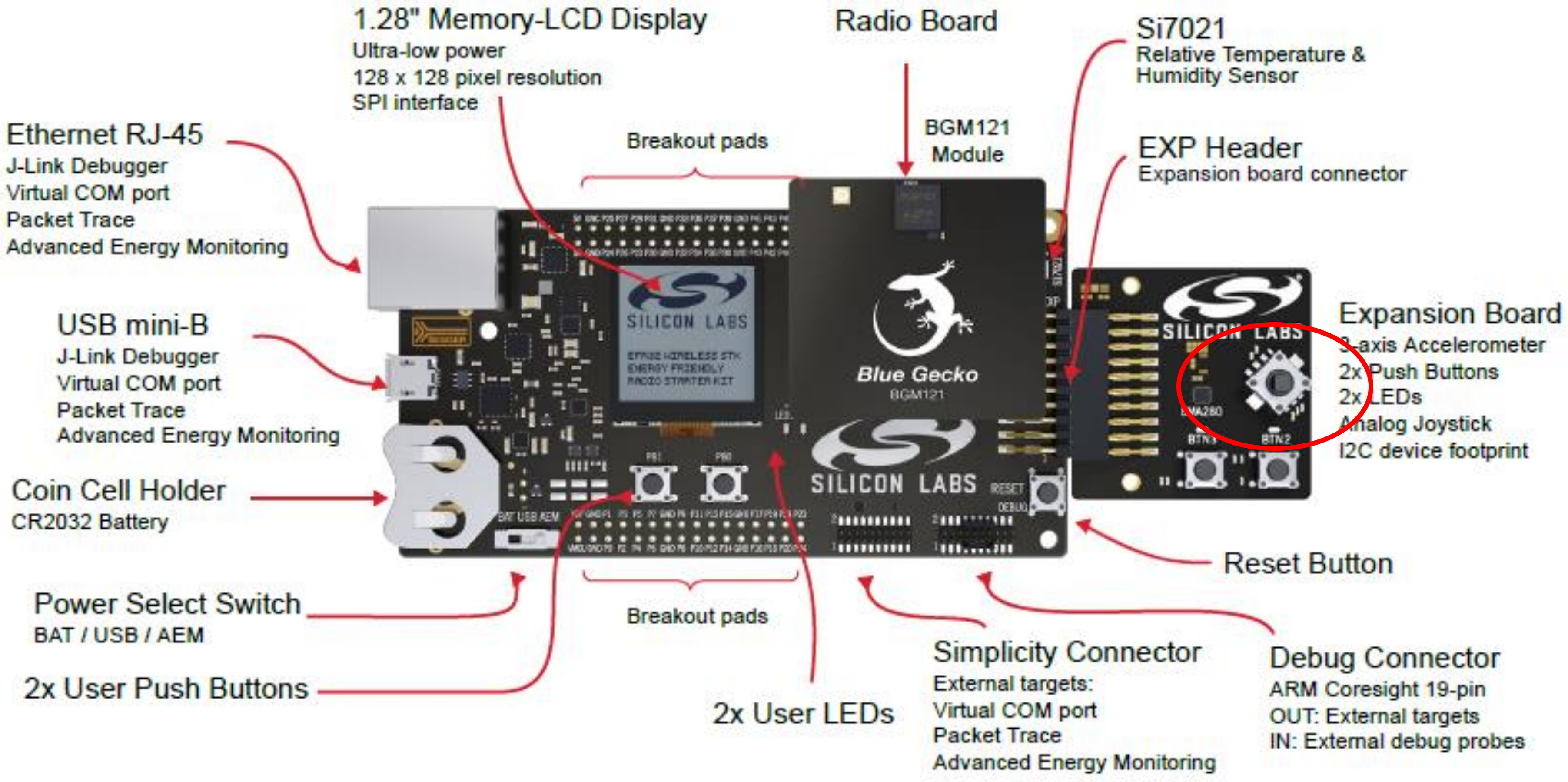
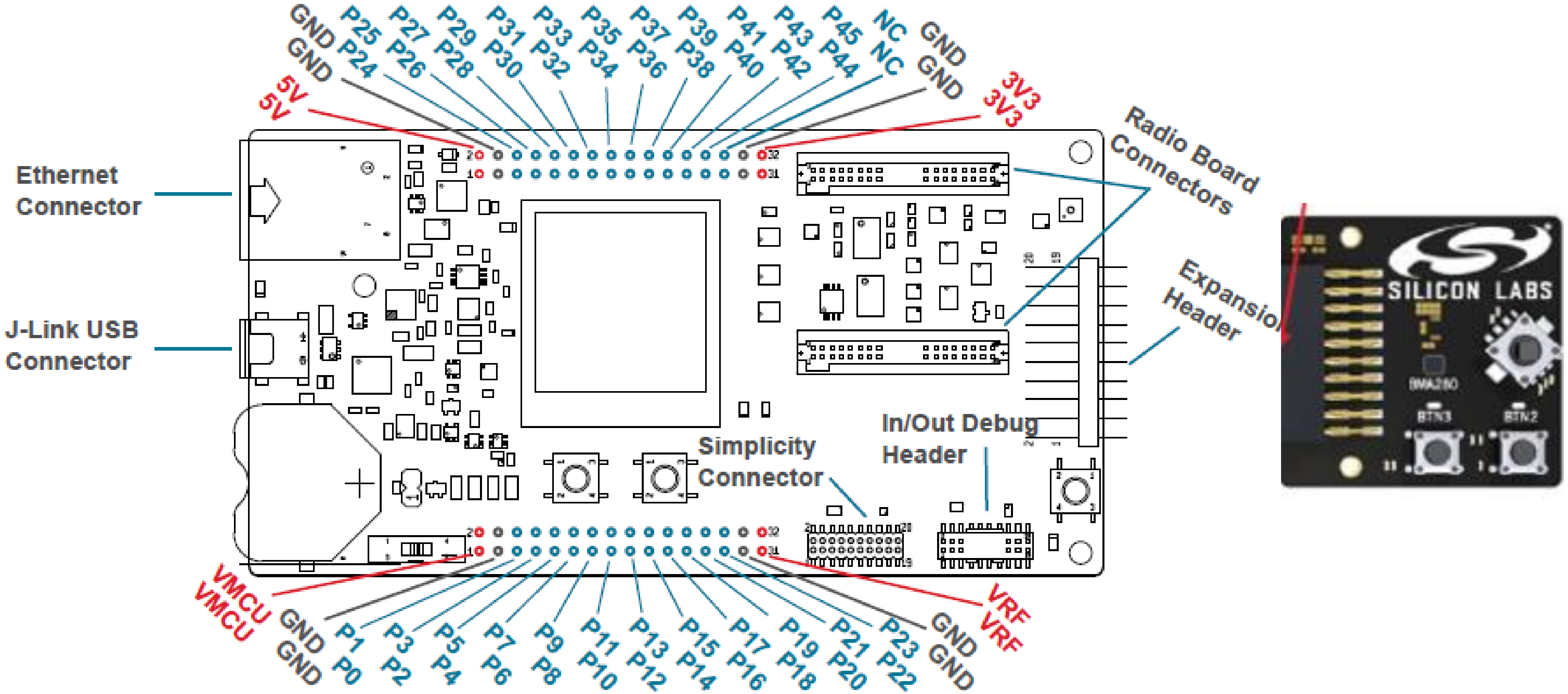


Figure 2.1. Kit Hardware Layout

ug234-brd4302a-user-guide

STK6101C – BGM121 Blue Gecko dev kit



GND P25 P27 P29 P31 P33 P35 P37 P39 P41 P43 P45 NC GND
GND P24 P26 P28 P30 P32 P34 P36 P38 P40 P42 P44 NC GND
5V
5V
3V3
3V3
Ethernet Connector
J-Link USB Connector
Radio Board Connectors
Expansio Header
Simplicity Connector
In/Out Debug Header
SILICON LABS
BMA280
BTN3
BTN2
VMCU GND P1 P3 P5 P7 P9 P11 P13 P15 P17 P19 P21 P23 GND VRF
VMCU GND P0 P2 P4 P6 P8 P10 P12 P14 P16 P18 P20 P22 GND VRF

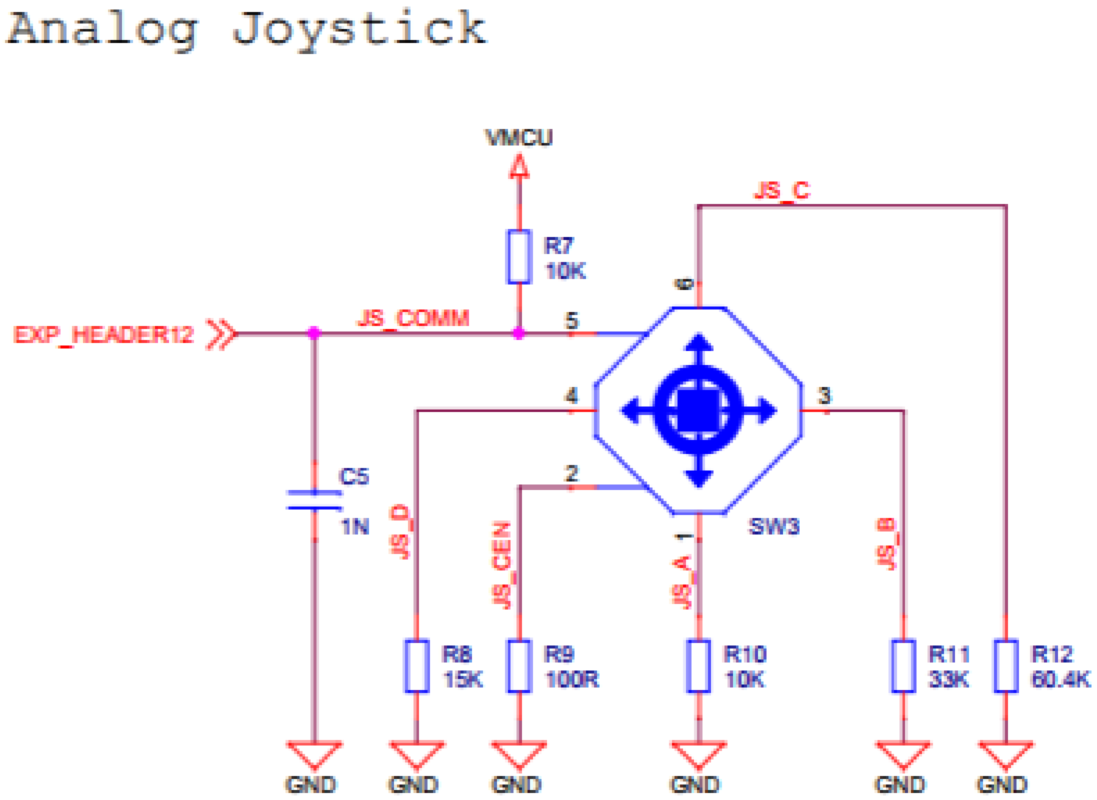
Figure 3.1. Mainboard Connector Layout

ug234-brd4302a-user-guide

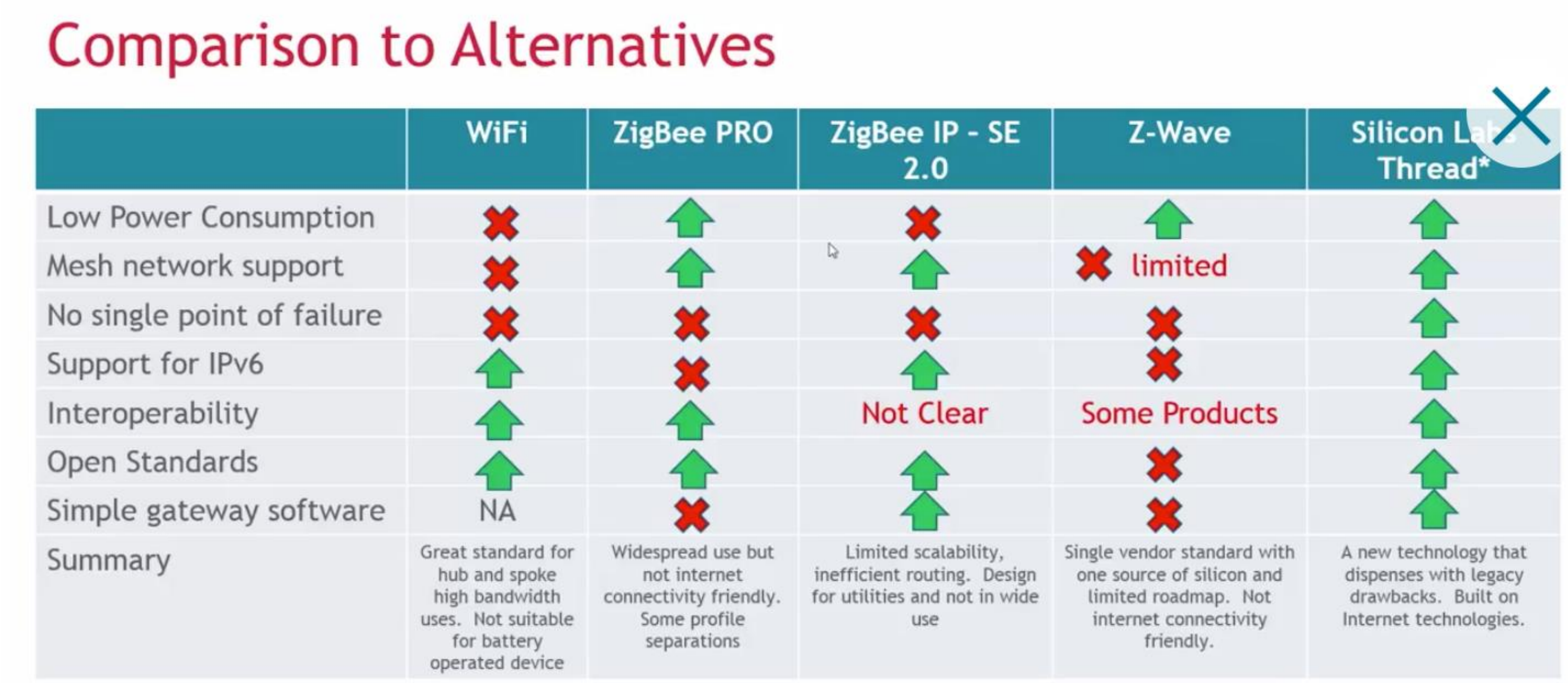
STK6101C – BGM121 Blue Gecko dev kit

What voltage would the Joy Stick output be if it is pulled down, south?

What voltage would the Joy Stick output be if it is depressed, pushed down?

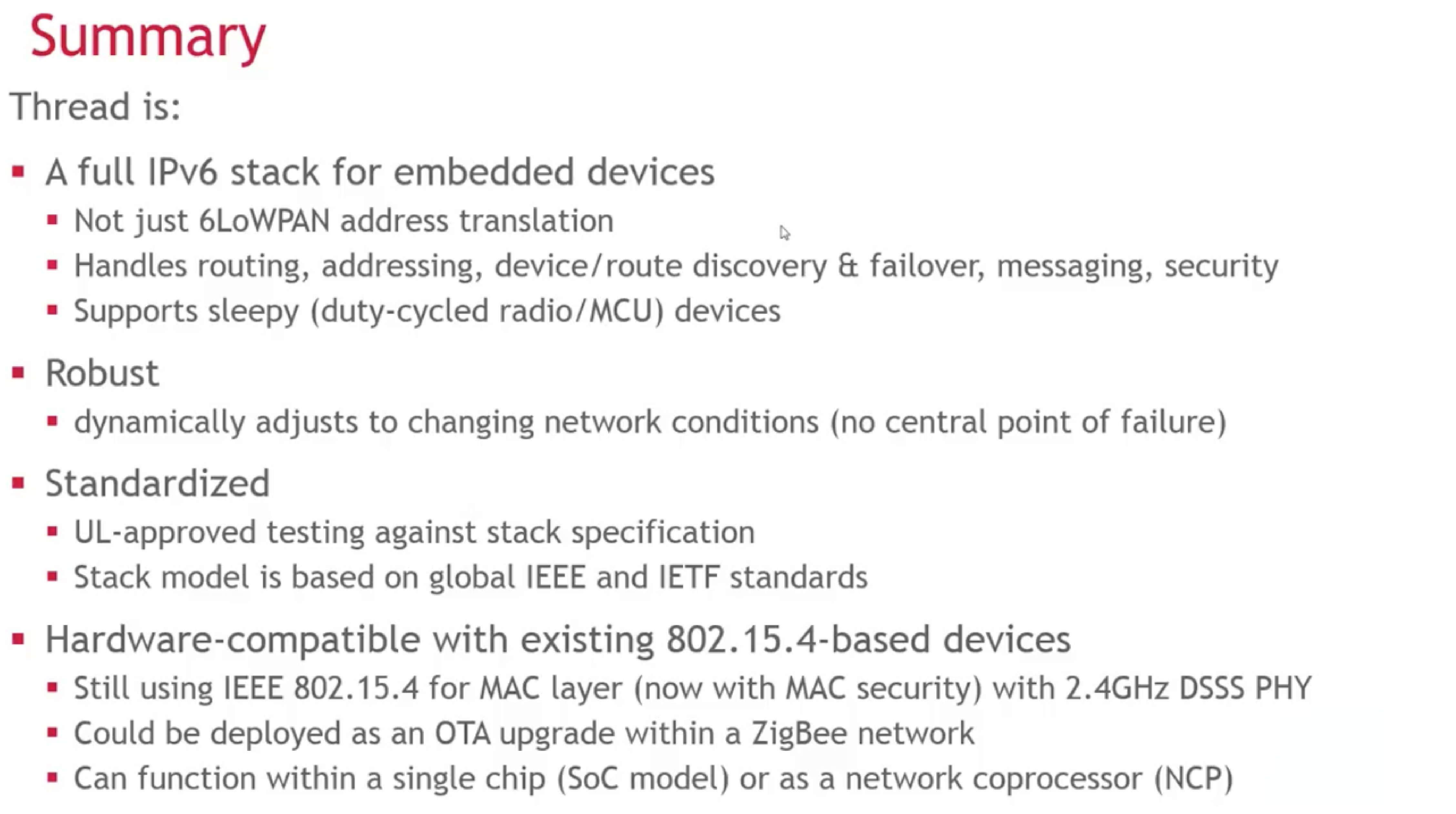


Wireless comparisons



Comparison to Alternatives

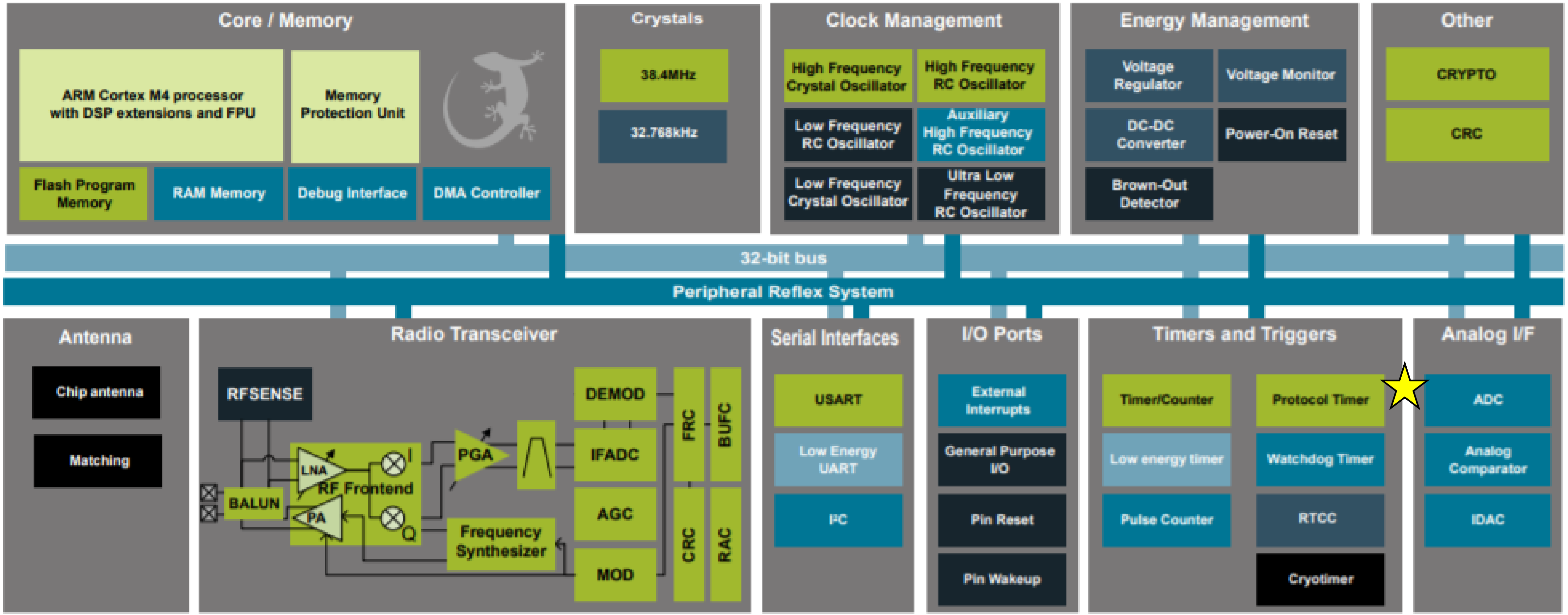
	WiFi	ZigBee PRO	ZigBee IP - SE 2.0	Z-Wave	Silicon Labs Thread*
Low Power Consumption	✖	⬆	✖	⬆	⬆
Mesh network support	✖	⬆	⬆	✖ limited	⬆
No single point of failure	✖	✖	✖	✖	⬆
Support for IPv6	⬆	✖	⬆	✖	⬆
Interoperability	⬆	⬆	Not Clear	Some Products	⬆
Open Standards	⬆	⬆	⬆	✖	⬆
Simple gateway software	NA	✖	⬆	✖	⬆
Summary	Great standard for hub and spoke high bandwidth uses. Not suitable for battery operated device	Widespread use but not internet connectivity friendly. Some profile separations	Limited scalability, inefficient routing. Design for utilities and not in wide use	Single vendor standard with one source of silicon and limited roadmap. Not internet connectivity friendly.	A new technology that dispenses with legacy drawbacks. Built on Internet technologies.



Summary

Thread is:

- A full IPv6 stack for embedded devices
  - Not just 6LoWPAN address translation
  - Handles routing, addressing, device/route discovery & failover, messaging, security
  - Supports sleepy (duty-cycled radio/MCU) devices
- Robust
  - dynamically adjusts to changing network conditions (no central point of failure)
- Standardized
  - UL-approved testing against stack specification
  - Stack model is based on global IEEE and IETF standards
- Hardware-compatible with existing 802.15.4-based devices
  - Still using IEEE 802.15.4 for MAC layer (now with MAC security) with 2.4GHz DSSS PHY
  - Could be deployed as an OTA upgrade within a ZigBee network
  - Can function within a single chip (SoC model) or as a network coprocessor (NCP)



Core / Memory

ARM Cortex M4 processor with DSP extensions and FPU

Memory Protection Unit

Flash Program Memory

RAM Memory

Debug Interface

DMA Controller

Crystals

38.4MHz

32.768kHz

Clock Management

High Frequency Crystal Oscillator

High Frequency RC Oscillator

Low Frequency RC Oscillator

Auxiliary High Frequency RC Oscillator

Low Frequency Crystal Oscillator

Ultra Low Frequency RC Oscillator

Energy Management

Voltage Regulator

Voltage Monitor

DC-DC Converter

Power-On Reset

Brown-Out Detector

Other

CRYPTO

CRC

32-bit bus

Peripheral Reflex System

Antenna

Chip antenna

Matching

Radio Transceiver

RFSENSE

BALUN

LNA

PA

RF Frontend

I

Q

PGA

Frequency Synthesizer

DEMOD

IFADC

AGC

MOD

FRC

BUFC

CRC

RAC

Serial Interfaces

USART

Low Energy UART

I^2C

I/O Ports

External Interrupts

General Purpose I/O

Pin Reset

Pin Wakeup

Timers and Triggers

Timer/Counter

Protocol Timer

Low energy timer

Watchdog Timer

Pulse Counter

RTCC

Cryotimer

Analog I/F

ADC

Analog Comparator

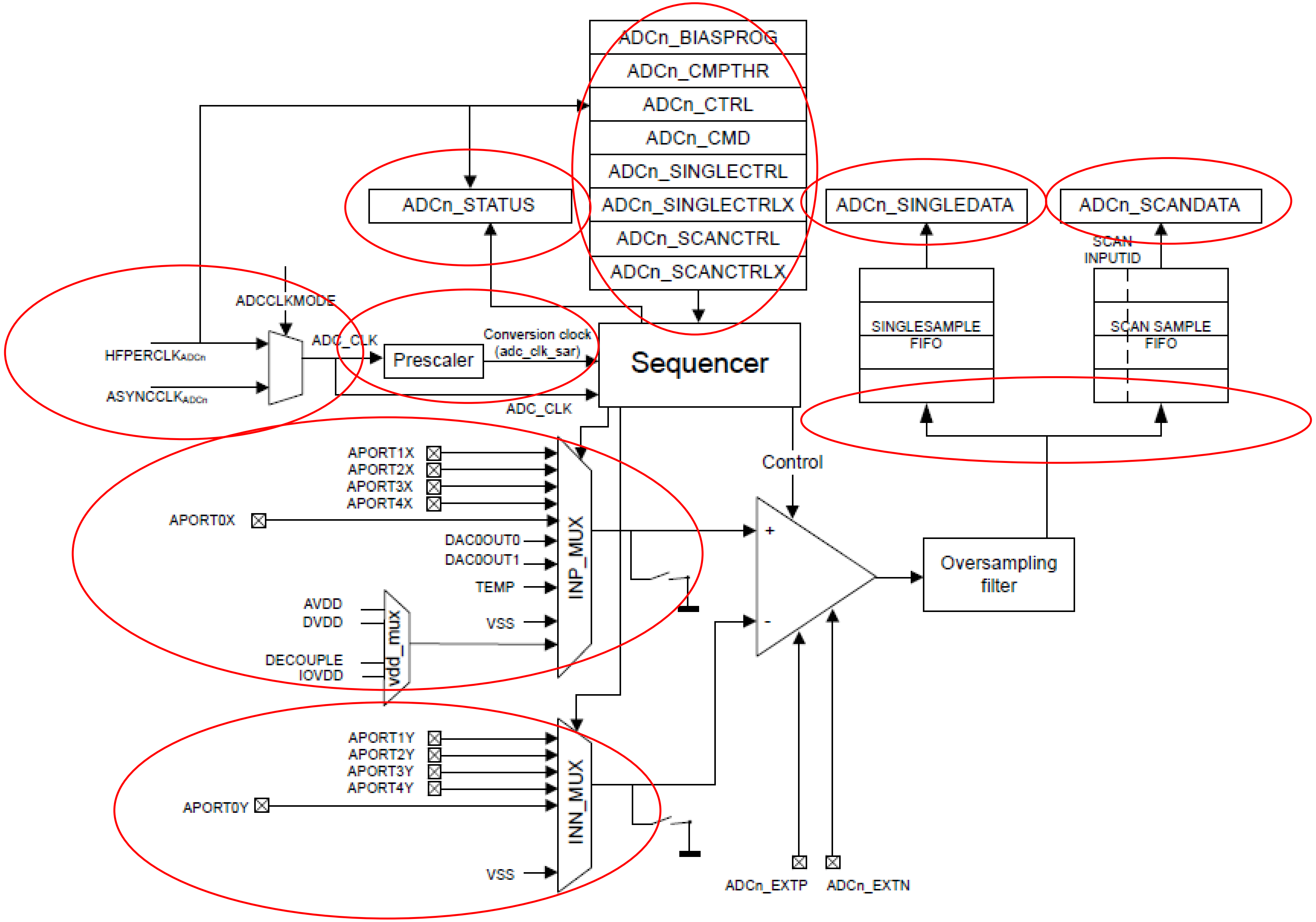
IDAC

Lowest power mode with peripheral operational:

EM0—Active | EM1—Sleep | EM2—Deep Sleep | EM3—Stop | EM4—Hibernate | EM4—Shutoff

Silicon Labs' EFR32BG Reference Manual

Analog Comparator (ADC0)

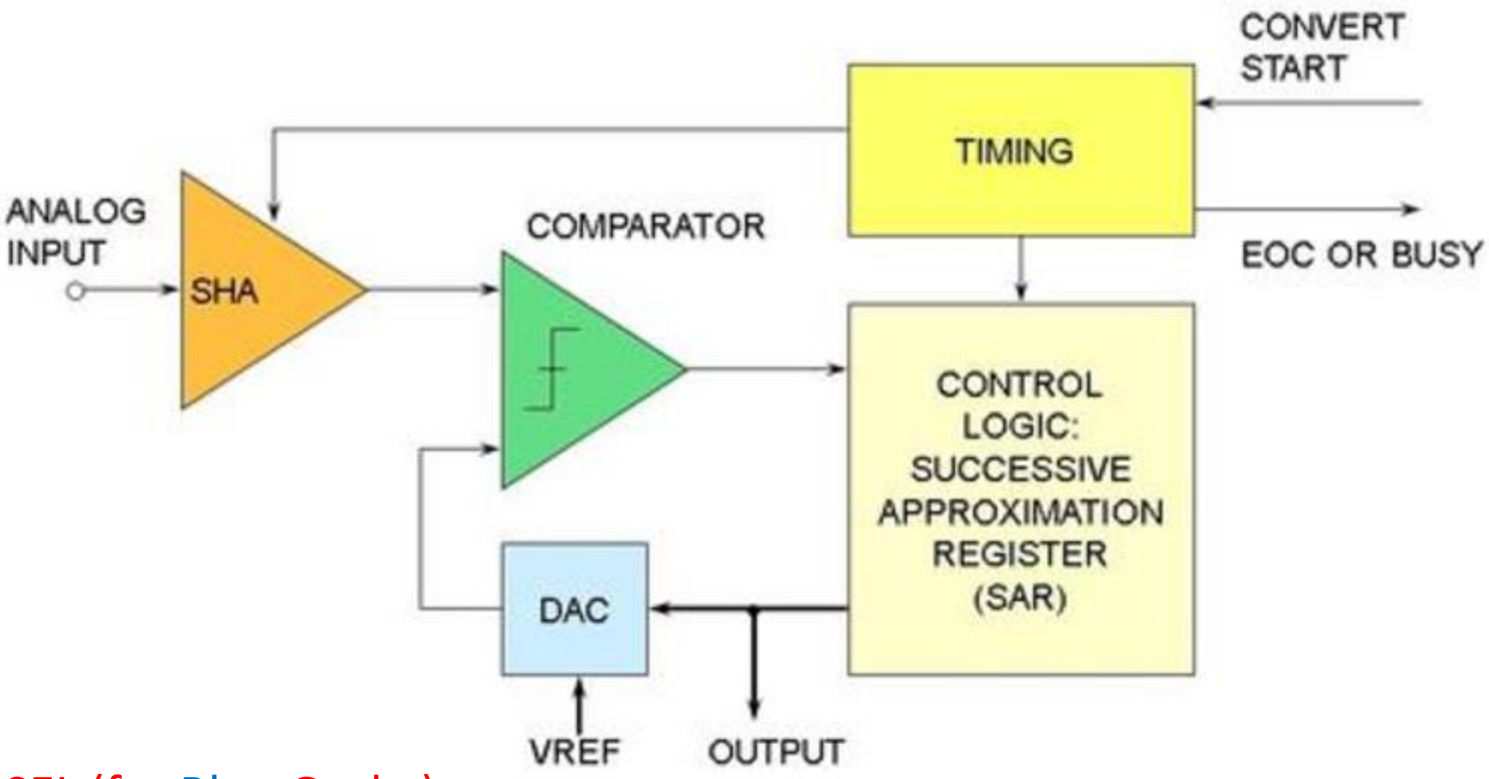


Silicon Labs' EFR32BG reference manual

SAR – Successive Approximation Register ADC

As shown in Figure 1a, the SAR ADC samples the input signal once at each convert start edge and compares bit on each clock edge. It then adjusts the output of the digital to analog converter (DAC) through control logic until the DAC output very closely matches the analog input. It requires N-number of clock cycles from an independent external clock to implement a single N-bit conversion in an iterative manner.

Figure 1a

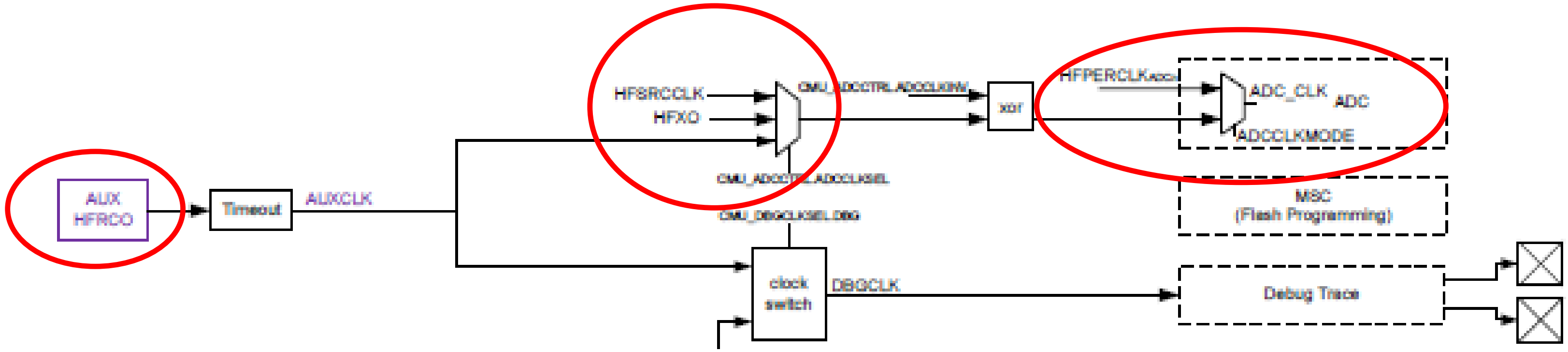


Basic SAR Architecture

Tconv= (Tacq+ (N + 1) x Tadc_clk_sar) x OVSRSEL (for Blue Gecko)

http://www.planetanalog.com/author.asp?section_id=3193&doc_id=562097

Setting up the ADC_CLK clock tree



- The ADC peripheral has a dedicated high frequency clock to enable the remaining high frequency clock tree to be disabled to save energy
- The AUX HFRCO frequency is generated via a separate source from HFXO or HFRCO, thus it is running ASYNC to the HF Clock tree
- Separate AUX HFRCO prescaler used to enable a full range of sampling rates and to minimize energy

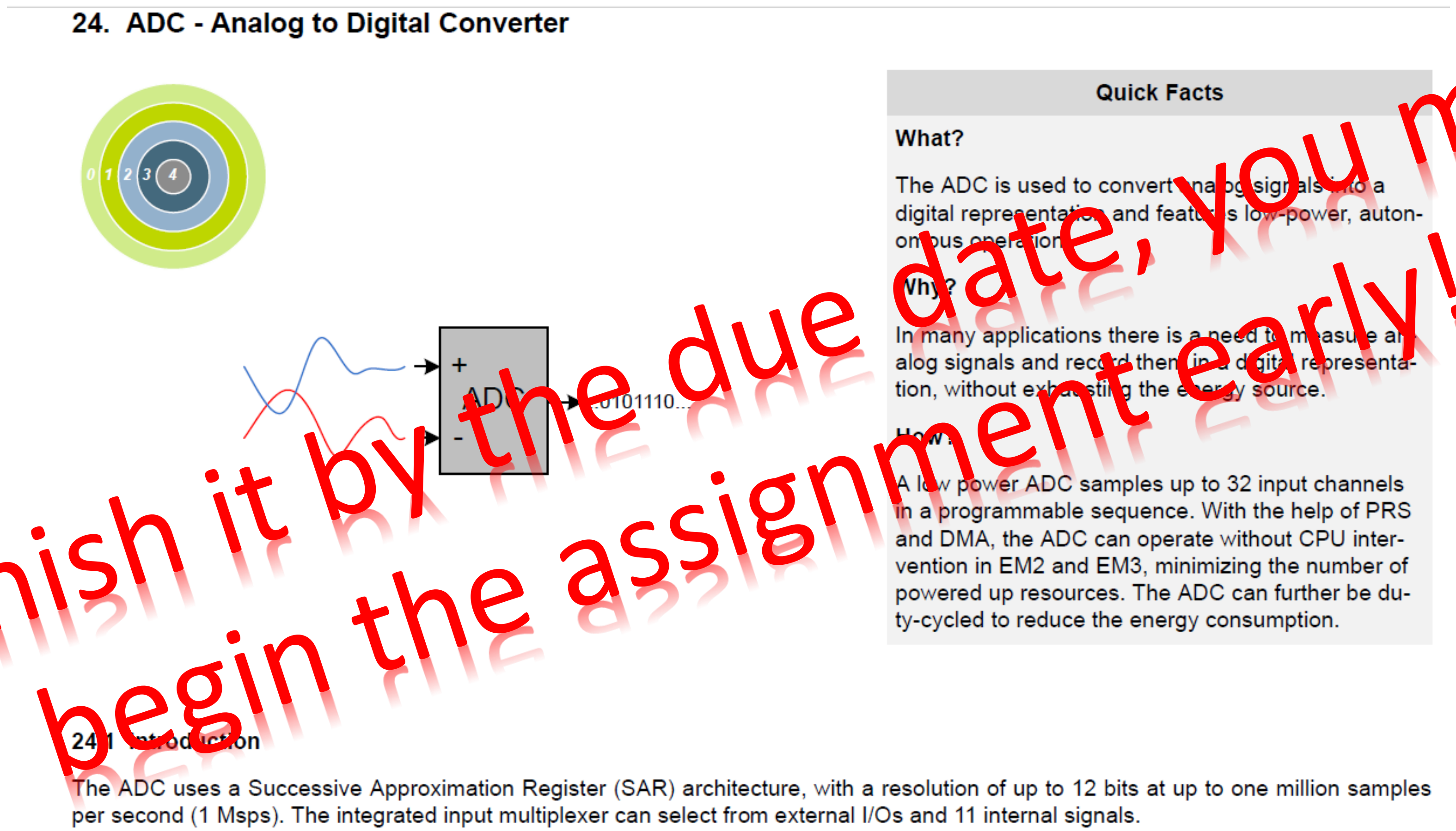
Silicon Labs' EFR32BG reference manual

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 as well as the ADC will not sample

Silicon Labs' EFR32BG reference manual

HIGHLY RECOMMENDED! READ the BG ADC Reference Manual Chapter



24. ADC - Analog to Digital Converter

Quick Facts

What?

The ADC is used to convert analog signals into a digital representation and features low-power, autonomous operation.

Why?

In many applications there is a need to measure analog signals and record them in a digital representation, without exhausting the energy source.

How?

A low power ADC samples up to 32 input channels in a programmable sequence. With the help of PRS and DMA, the ADC can operate without CPU intervention in EM2 and EM3, minimizing the number of powered up resources. The ADC can further be duty-cycled to reduce the energy consumption.

24.1 Introduction

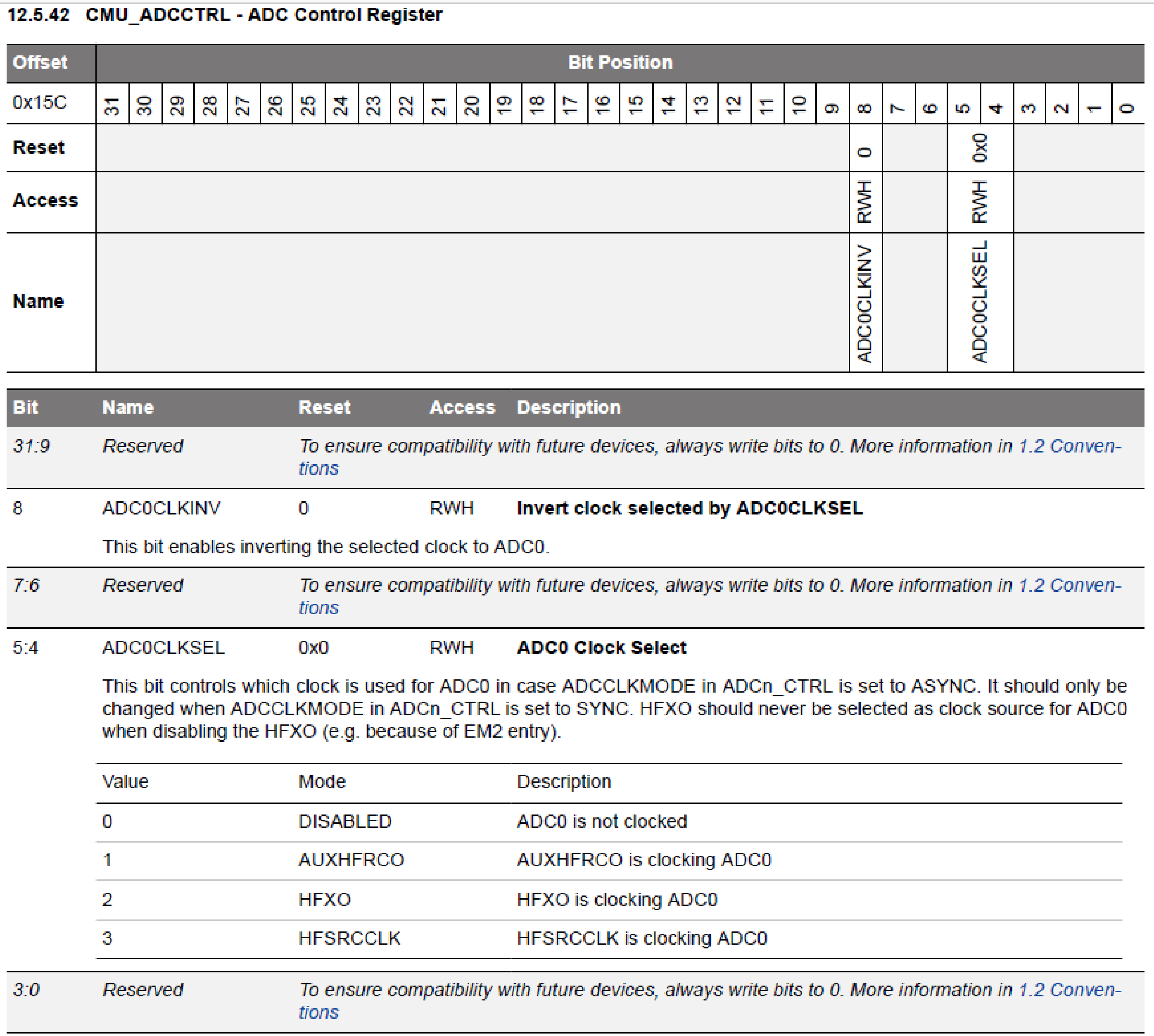
The ADC uses a Successive Approximation Register (SAR) architecture, with a resolution of up to 12 bits at up to one million samples per second (1 Msps). The integrated input multiplexer can select from external I/Os and 11 internal signals.

To finish it by the due date, you must begin the assignment early!

Silicon Labs' EFR32BG reference manual

Setting up the ADC

- Key ADC CMU clocking registers
  - CMU->ADCCTRL – Setting the clock source to the ADC
  - ADCn clock source can be specified as the ASYNC or HFPERCLK clock source
  - Will need to write to register directly to set up ADC clock source



12.5.42 CMU_ADCCTRL - ADC Control Register

Offset	Bit Position																															
0x15C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								0			0x0					
Access																								RWH			RWH					
Name																								ADC0CLKINV			ADC0CLKSEL					

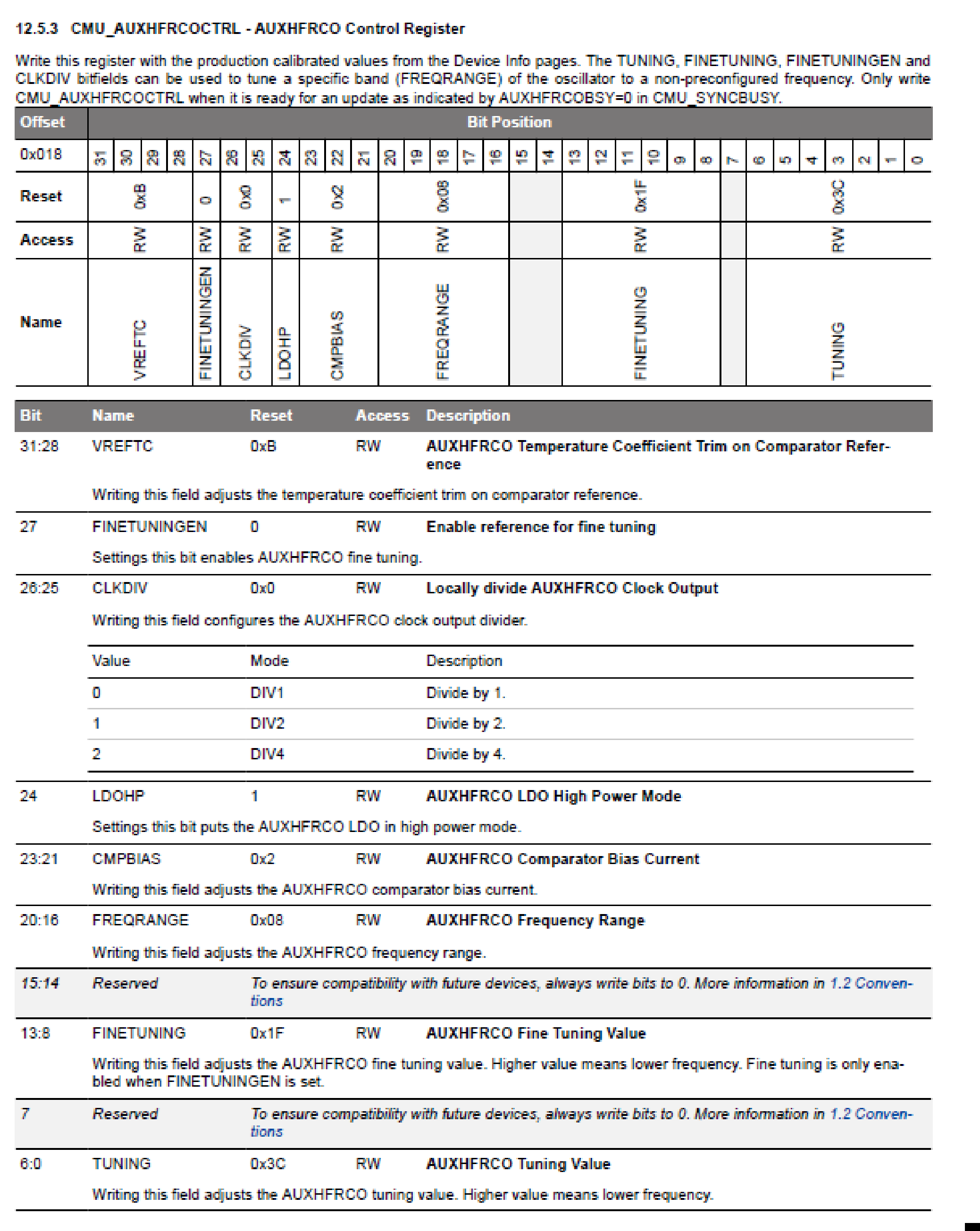
Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		
8	ADC0CLKINV	0	RWH	**Invert clock selected by ADC0CLKSEL** This bit enables inverting the selected clock to ADC0.
7:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		
5:4	ADC0CLKSEL	0x0	RWH	**ADC0 Clock Select** This bit controls which clock is used for ADC0 in case ADCCLKMODE in ADCn_CTRL is set to ASYNC. It should only be changed when ADCCLKMODE in ADCn_CTRL is set to SYNC. HFXO should never be selected as clock source for ADC0 when disabling the HFXO (e.g. because of EM2 entry).
3:0	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		

Value	Mode	Description
0	DISABLED	ADC0 is not clocked
1	AUXHFRCO	AUXHFRCO is clocking ADC0
2	HFXO	HFXO is clocking ADC0
3	HFSRCCLK	HFSRCCLK is clocking ADC0

Silicon Labs' EFR32BG reference manual

Setting up the ADC

- Key ADC CMU clocking registers
  - CMU->AUXHFRCOCTRL – Setting the clock frequency of AUXFRCO if required
  - **CMU_AUXHFRCOBandSet(*cmuAUXHFRCOFreq_xxHz*);**
  - **CMU_OscillatorEnable(*cmuOsc_AUXHFRCO, true, true*);**



12.5.3 CMU_AUXHFRCOCTRL - AUXHFRCO Control Register

Write this register with the production calibrated values from the Device Info pages. The TUNING, FINETUNING, FINETUNINGEN and CLKDIV bitfields can be used to tune a specific band (FREQRANGE) of the oscillator to a non-preconfigured frequency. Only write CMU_AUXHFRCOCTRL when it is ready for an update as indicated by AUXHFRCOBSY=0 in CMU_SYNCBUSY.

Offset	Bit Position									
0x018	31 30 29 28	27	26 25	24	23 22 21	20 19 18 17 16	15 14	13 12 11 10 9 8	7	6 5 4 3 2 1 0
Reset	0xB	0	0x0	1	0x2	0x08		0x1F		0x3C
Access	RW	RW	RW	RW	RW	RW		RW		RW
Name	VREFTC	FINETUNINGEN	CLKDIV	LDOHP	CMPBIAS	FREQRANGE		FINETUNING		TUNING

Bit	Name	Reset	Access	Description
31:28	VREFTC	0xB	RW	**AUXHFRCO Temperature Coefficient Trim on Comparator Reference** Writing this field adjusts the temperature coefficient trim on comparator reference.
27	FINETUNINGEN	0	RW	**Enable reference for fine tuning** Settings this bit enables AUXHFRCO fine tuning.
26:25	CLKDIV	0x0	RW	**Locally divide AUXHFRCO Clock Output** Writing this field configures the AUXHFRCO clock output divider.
24	LDOHP	1	RW	**AUXHFRCO LDO High Power Mode** Settings this bit puts the AUXHFRCO LDO in high power mode.
23:21	CMPBIAS	0x2	RW	**AUXHFRCO Comparator Bias Current** Writing this field adjusts the AUXHFRCO comparator bias current.
20:16	FREQRANGE	0x08	RW	**AUXHFRCO Frequency Range** Writing this field adjusts the AUXHFRCO frequency range.
15:14	*Reserved*			*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*
13:8	FINETUNING	0x1F	RW	**AUXHFRCO Fine Tuning Value** Writing this field adjusts the AUXHFRCO fine tuning value. Higher value means lower frequency. Fine tuning is only enabled when FINETUNINGEN is set.
7	*Reserved*			*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*
6:0	TUNING	0x3C	RW	**AUXHFRCO Tuning Value** Writing this field adjusts the AUXHFRCO tuning value. Higher value means lower frequency.

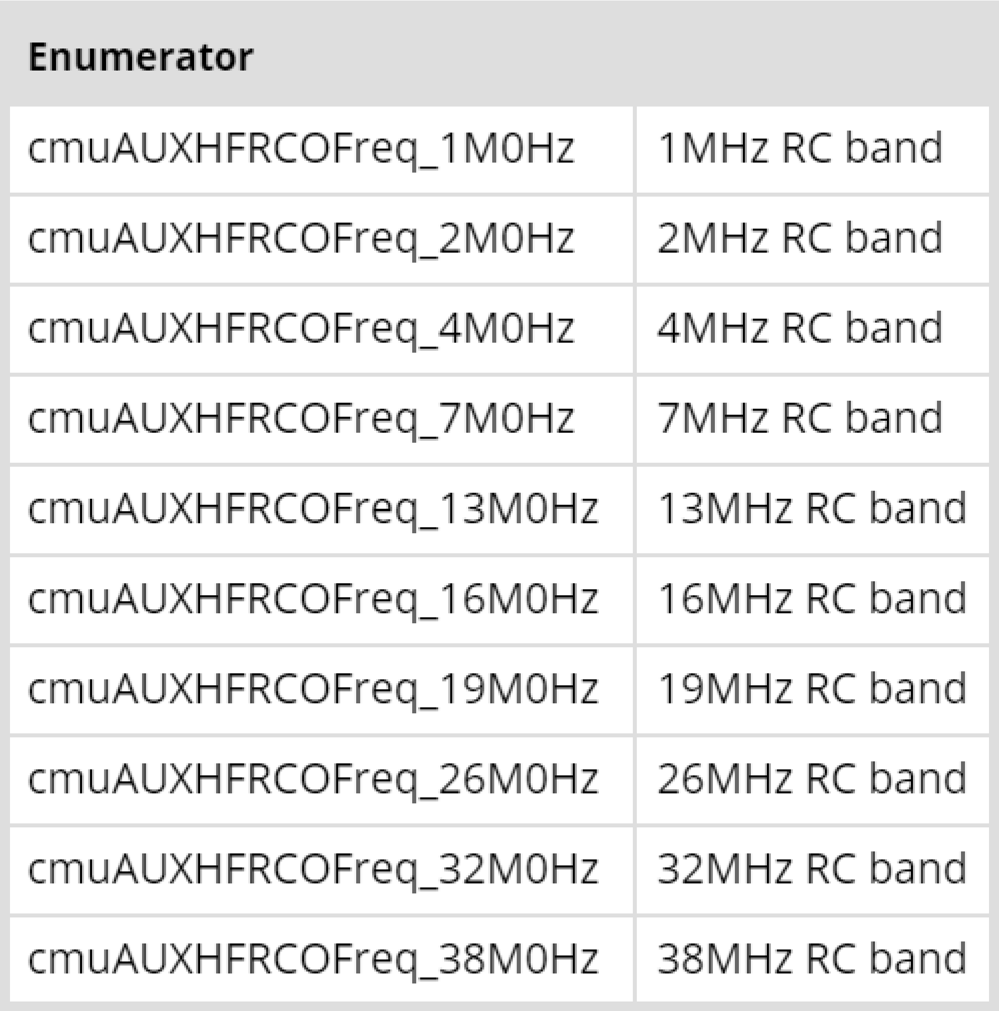
CLKDIV (26:25):

Value	Mode	Description
0	DIV1	Divide by 1.
1	DIV2	Divide by 2.
2	DIV4	Divide by 4.

Silicon Labs' EFR32BG reference manual

Setting up the ADC

- Note that the maximum clock frequency for adc_clk_sar is 16 MHz. The ADC warmup time is determined by ADC_CLK and not by adc_clk_sar
  - IF AUX HFRCO is > 16 MHz, the prescaler must be greater than 1

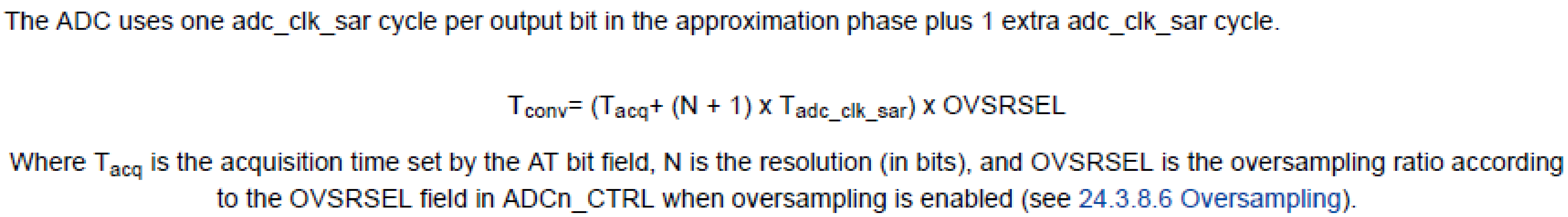


Enumerator	
cmuAUXHFRCOFreq_1M0Hz	1MHz RC band
cmuAUXHFRCOFreq_2M0Hz	2MHz RC band
cmuAUXHFRCOFreq_4M0Hz	4MHz RC band
cmuAUXHFRCOFreq_7M0Hz	7MHz RC band
cmuAUXHFRCOFreq_13M0Hz	13MHz RC band
cmuAUXHFRCOFreq_16M0Hz	16MHz RC band
cmuAUXHFRCOFreq_19M0Hz	19MHz RC band
cmuAUXHFRCOFreq_26M0Hz	26MHz RC band
cmuAUXHFRCOFreq_32M0Hz	32MHz RC band
cmuAUXHFRCOFreq_38M0Hz	38MHz RC band

Silicon Labs' EFR32BG reference manual

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 requires one clock cycle per output bit in the approximation phase + 1 extra ADC.



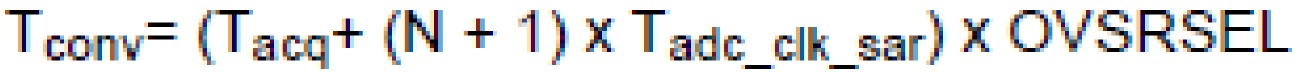
The ADC uses one adc_clk_sar cycle per output bit in the approximation phase plus 1 extra adc_clk_sar cycle.

$$T_{conv} = (T_{acq} + (N + 1) \times T_{adc\_clk\_sar}) \times OVSRSEL$$

Where T_{acq} is the acquisition time set by the AT bit field, N is the resolution (in bits), and OVSRSEL is the oversampling ratio according to the OVSRSEL field in ADCn_CTRL when oversampling is enabled (see 24.3.8.6 Oversampling).

Silicon Labs’ EFR32BG reference manual

Calculating MSPS



$$T_{conv} = (T_{acq} + (N + 1) \text{ x } T_{adc\_clk\_sar}) \text{ x OVSRSEL}$$

- HFRCO = 19 MHz
- Prescalar = 4
- 12 bit conversion
- Acquisition time > 3uS
- No Oversampling => OSR = 1
- ADC_clock = 19MHz/4 = 4.75MHz
- Acquisition time
  - $3\text{x}10^{-6} * (4.75\text{x}10^{6})$ = 10.5 clock cycles
  - T_A = 14.25 clock cycles => 16
- 12-bit conversion
  - N = 12

- $T_{conv} = (T_A + N + 1) \text{ x OSR}$
- $T_{conv} = (16+12+1) \text{ x } 1$
- T_{conv} = 29 ADC_Clocks
- $T_{conv} = 29 \text{ x } (1/4.75\text{x}10^{6})$
- T_{conv} = 6.1uS
- MSPS = 1/ T_{conv}
- MSPS = 1/6.1uS
- MSPS = 0.164 or 164 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
- ADC warm-up required is 5 times the time base, 5 x 1uS = 5uS, if kept “warmed,” the warm-up time equals the time base of 1uS
- 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: This is the lowest power option for general-purpose use and low sampling rates (below 35 ksps)
  - KEEPINSTANDBY: It may also be useful for lower sampling rates where latency is important.
  - KEEPINSLOWACC: This mode is useful for high-impedance inputs which are sampled infrequently
  - KEEPADCWARM: This mode provides the lowest latency and allows for maximum sampling rates

Silicon Labs’ EFR32BG reference manual

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, not ADC Clocks, which corresponds to at least 1 µs.
  - To set the Time base, the following library routine can be used:
    - ADC_TimebaseCalc(0);

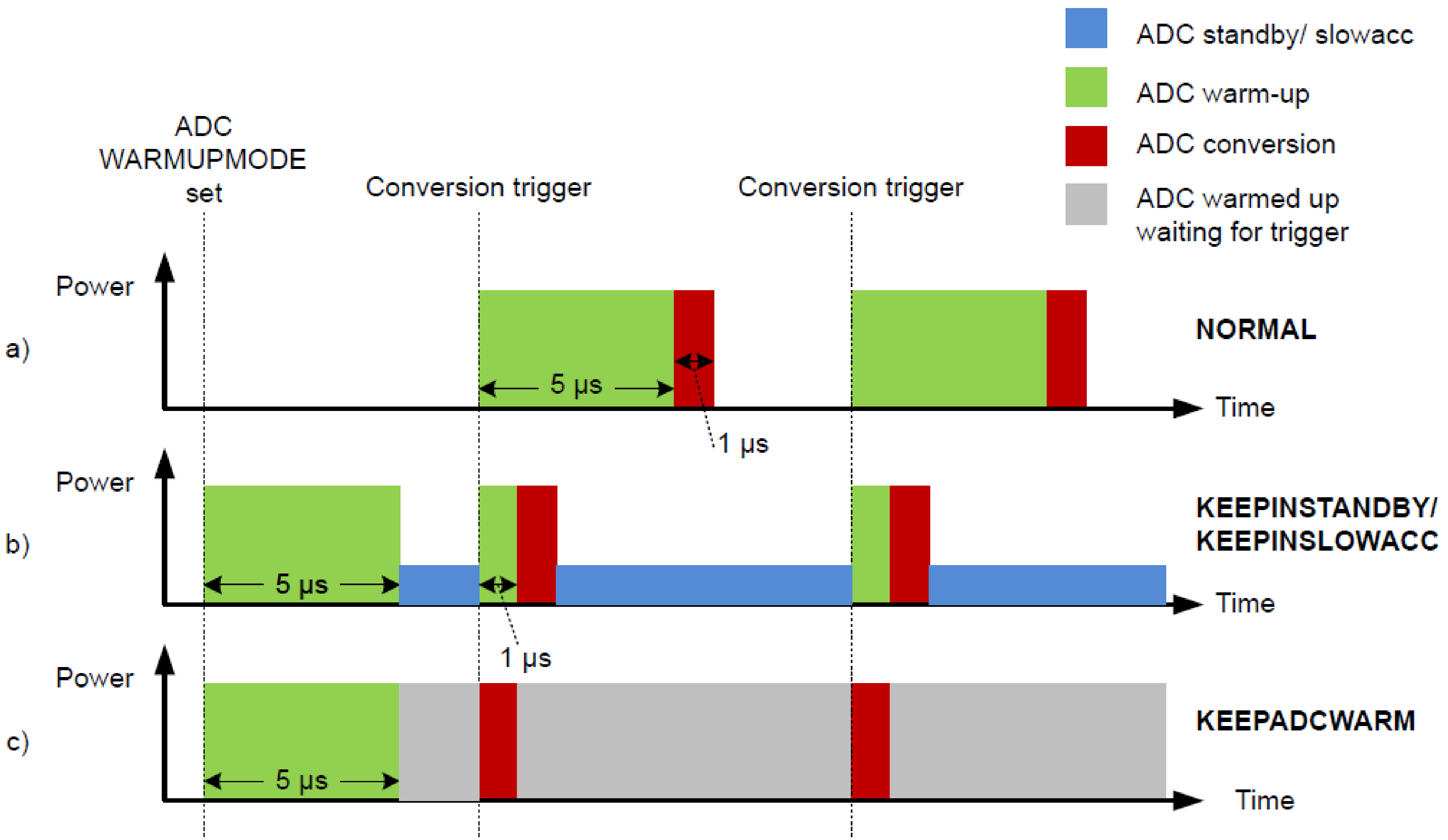


Figure 24.4. ADC Analog Power Consumption With Different WARMUPMODE Settings

Silicon Labs' EFR32BG reference manual

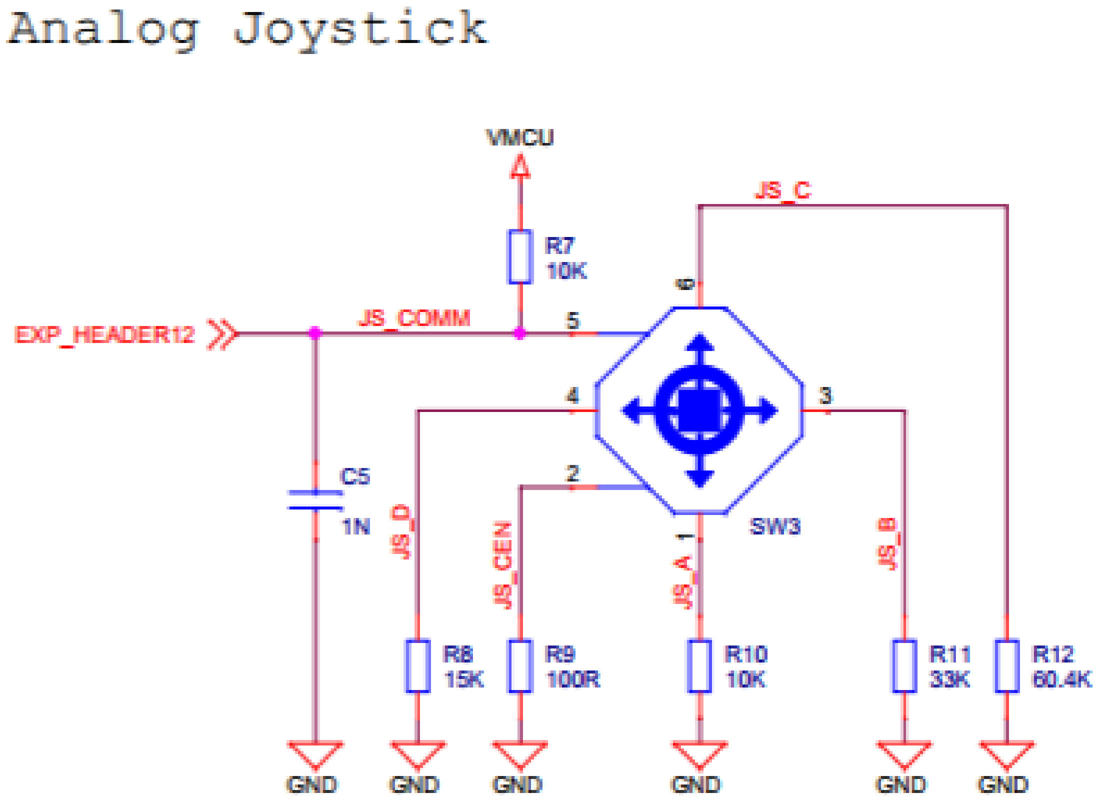
ADC – Reference Selection

- Programmable and preset input full scale (peak-to-peak) range (VFS) with selectable reference sources
  - VFS=1.25 V using internal VBGR reference
  - VFS=2.5 V using internal VBGR reference
  - VFS=AVDD with AVDD as reference source
  - VFS=5 V with internal VBGR reference
  - Single ended external reference
  - Differential external reference
  - VFS=2xAVDD with AVDD as reference source
  - User-programmable dividers for flexible VFS options from internal, external or supply voltage reference sources

Silicon Labs’ EFR32BG1 data sheet

STK6101C – BGM121 Blue Gecko dev kit

Should you be using the ADC in Single Ended or Differential mode?



ADC – Reference Selection

- **24.3.6 Reference Selection and Input Range Definition**
  - The full scale voltage (VFS) of the ADC is defined as the full input range, from the lowest possible input voltage to the highest. For single-ended conversions, the input range on the selected positive input is from 0 to VFS. For differential conversions, the input to the converter is the difference between the positive and negative input selections. This can range from -VFS/2 to +VFS/2.
  - VFS for the converter is determined by a combination of the selected voltage reference (VREF) and programmable divider circuits on the ADC input and voltage reference paths. Users have full control over the VREF and divider selections, offering a very flexible and wide selection of VFS values. In most applications however, it is not necessary to adjust VFS beyond a set of common pre-defined choices. For the simplest VFS configuration, refer to 24.3.6.1 Basic Full-Scale Voltage Configuration. If the application requires a VFS configuration not available in the pre-defined choices, 24.3.6.2 Advanced Full-Scale Voltage Configuration covers additional configuration options.

Silicon Labs' EFR32BG1 data sheet

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.

Silicon Labs' EFR32BG data sheet

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

Silicon Labs' EFR32BG data sheet

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 Blue Gecko enables Oversampling from 2 to 4096.
- Effectively increasing the resolution of the 12-bit ADC peripheral to 16 bits!

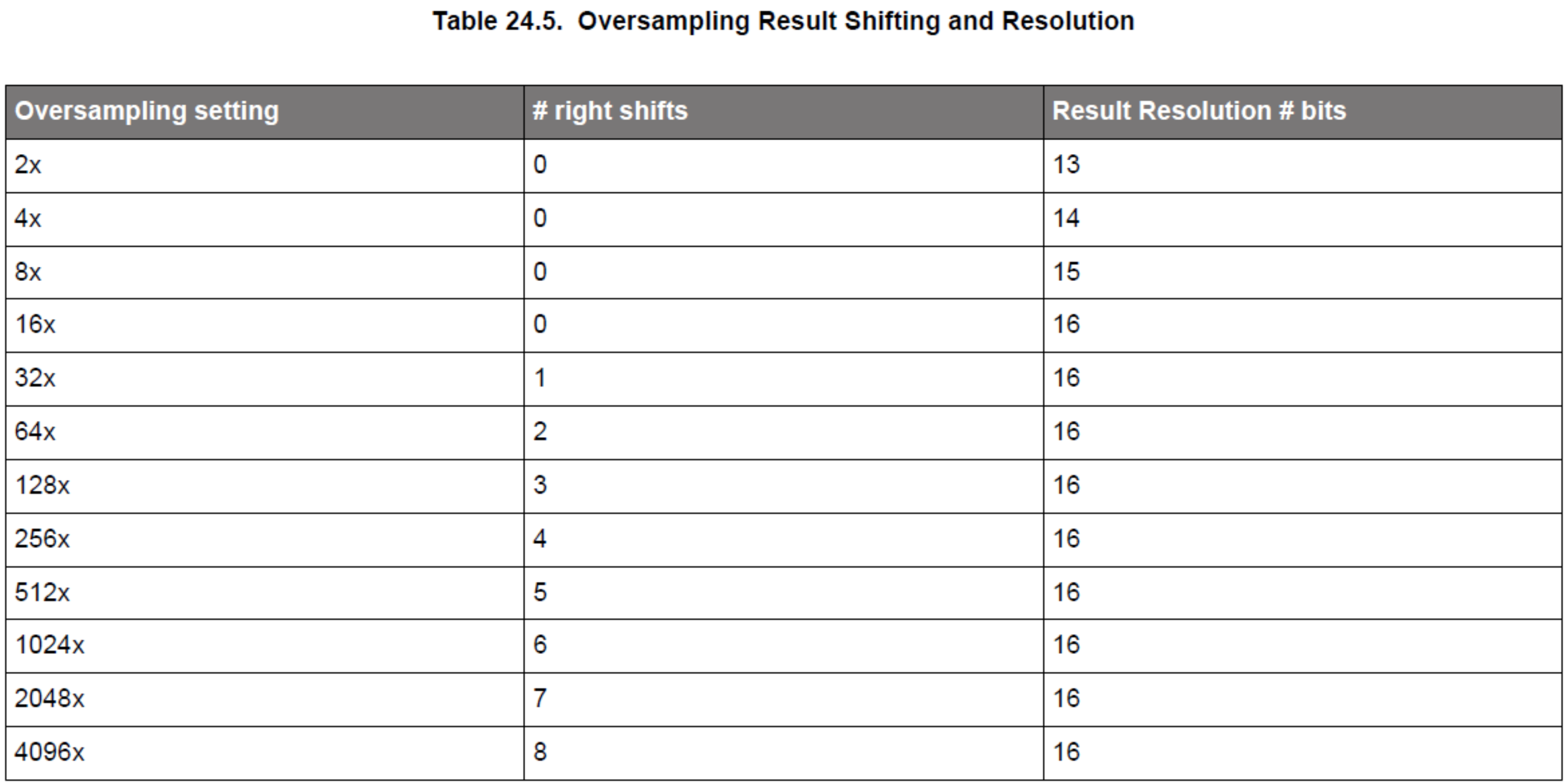


Table 24.5. Oversampling Result Shifting and Resolution

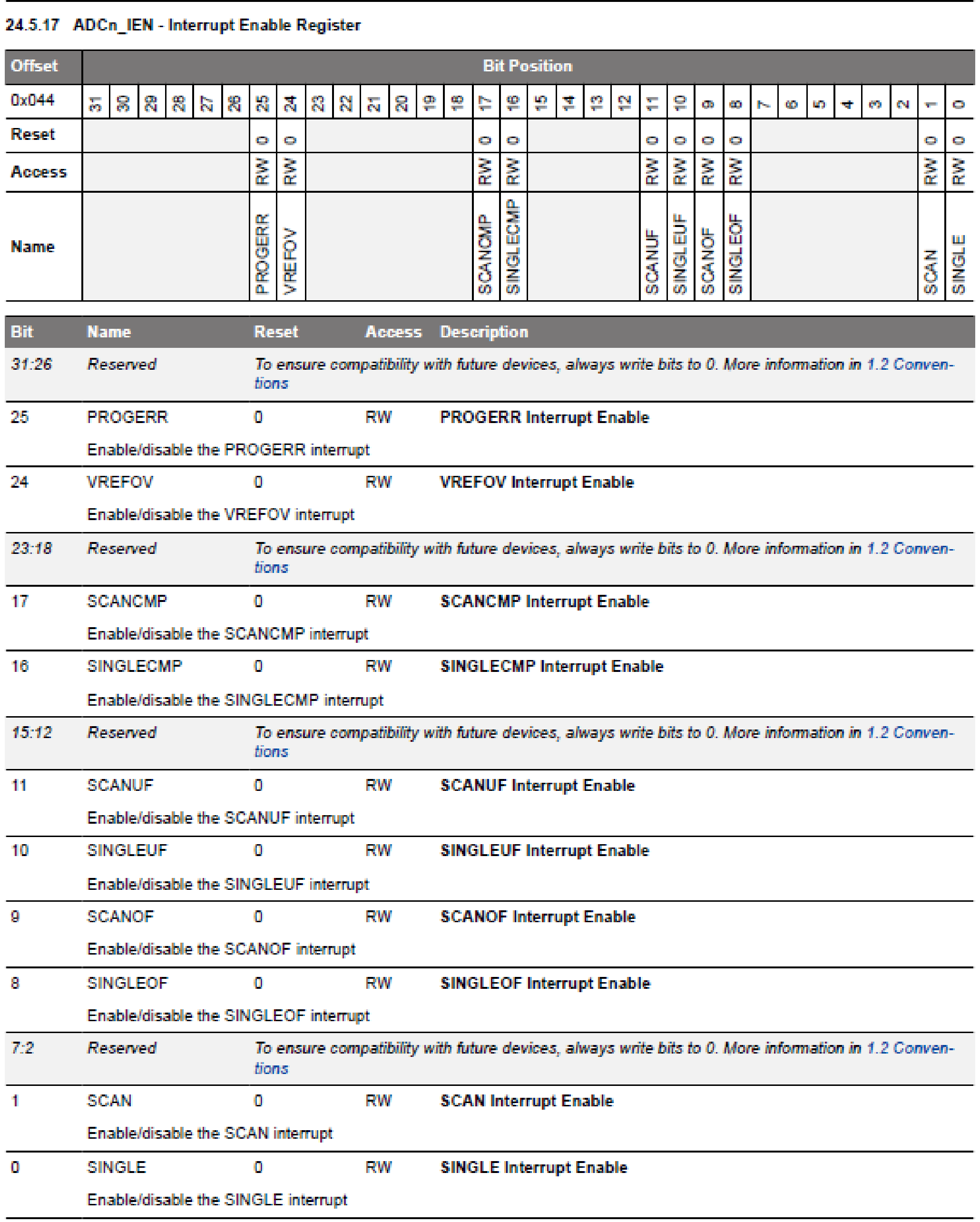
Oversampling setting	# right shifts	Result Resolution # bits
2x	0	13
4x	0	14
8x	0	15
16x	0	16
32x	1	16
64x	2	16
128x	3	16
256x	4	16
512x	5	16
1024x	6	16
2048x	7	16
4096x	8	16

Silicon Labs' EFR32BG1 data sheet

ADC – Interrupts

For this assignment, waiting for input from the analog Joy Stick, what would be the best interrupt to use for low energy?

- SINGLECMP - The ADC can be configured to window compare function to trip when the result reaches a certain threshold while gathering ADC data in EM2 or EM3.



24.5.17 ADCn_IEN - Interrupt Enable Register

Offset	Bit Position																															
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset							0	0							0	0					0	0	0	0							0	0
Access							RW	RW							RW	RW					RW	RW	RW	RW							RW	RW
Name							PROGERR	VREFOV							SCANCMP	SINGLECMP					SCANUF	SINGLEUF	SCANOF	SINGLEOF							SCAN	SINGLE

Bit	Name	Reset	Access	Description
31:26	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		
25	PROGERR	0	RW	**PROGERR Interrupt Enable**
	Enable/disable the PROGERR interrupt			
24	VREFOV	0	RW	**VREFOV Interrupt Enable**
	Enable/disable the VREFOV interrupt			
23:18	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		
17	SCANCMP	0	RW	**SCANCMP Interrupt Enable**
	Enable/disable the SCANCMP interrupt			
16	SINGLECMP	0	RW	**SINGLECMP Interrupt Enable**
	Enable/disable the SINGLECMP interrupt			
15:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		
11	SCANUF	0	RW	**SCANUF Interrupt Enable**
	Enable/disable the SCANUF interrupt			
10	SINGLEUF	0	RW	**SINGLEUF Interrupt Enable**
	Enable/disable the SINGLEUF interrupt			
9	SCANOF	0	RW	**SCANOF Interrupt Enable**
	Enable/disable the SCANOF interrupt			
8	SINGLEOF	0	RW	**SINGLEOF Interrupt Enable**
	Enable/disable the SINGLEOF interrupt			
7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		
1	SCAN	0	RW	**SCAN Interrupt Enable**
	Enable/disable the SCAN interrupt			
0	SINGLE	0	RW	**SINGLE Interrupt Enable**
	Enable/disable the SINGLE interrupt			

Silicon Labs' EFR32BG1 Reference Manual

ADC – Interrupts (Window Compare Function)

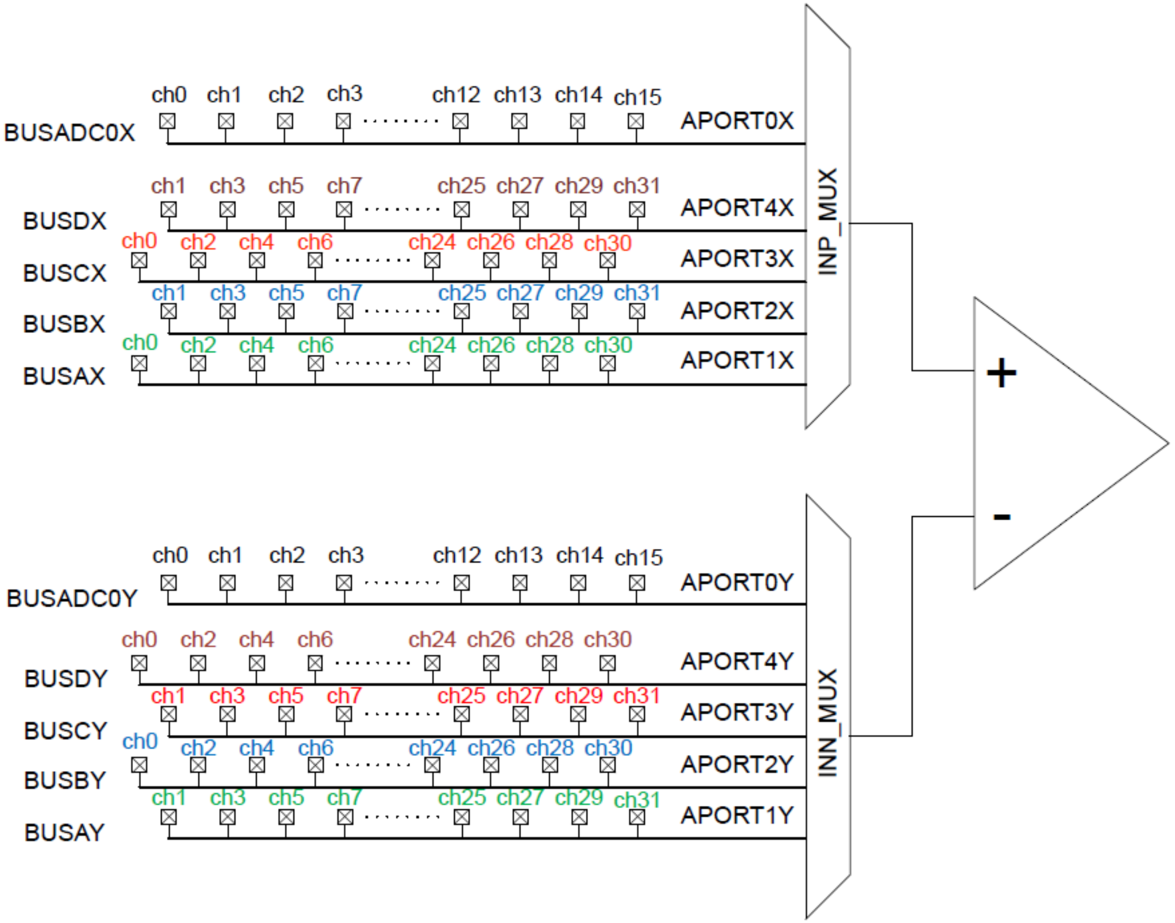
- The compare thresholds, ADGT and ADLT, are defined in the ADCn_CMPTHR register. These are 16-bit values and their format must match the type of conversion (single-ended or differential)
- There is a single set of ADLT and ADGT threshold for both single and scan compare
  - The user can however enable single or scan compare logic individually by enabling CMPEN in ADCn_SINGLECTRL or ADCn_SCANCTRL register
- The user can perform comparison both within or outside of the window defined by the ADGT and ADLT
  - If the ADLT is greater than ADGT, the ADC compares if the current sample is within the window.
  - If the ADLT is less than ADGT, the ADC compares if the current sample is outside of the window.

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

ADC – Selecting inputs

- The analog ports APORT1, APORT2, APORT3, and APORT4 connect to external pins via analog buses (BUSAX, BUSAY, BUSBX, etc.) which are shared among other analog peripherals on the device
- APORT1 through APORT4 are each 32 channels wide with connections to two sub-buses: a 16-channel X bus and a 16-channel Y bus. In the ADC module, all X buses connect to the INP_MUX and all Y buses connect to the INN_MUX as

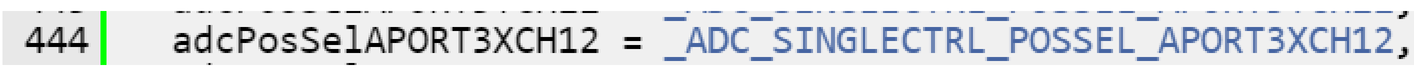


BUSADC0X
ch0 ch1 ch2 ch3 ch12 ch13 ch14 ch15
APORT0X
BUSDX
ch1 ch3 ch5 ch7 ch25 ch27 ch29 ch31
APORT4X
BUSCX
ch0 ch2 ch4 ch6 ch24 ch26 ch28 ch30
APORT3X
BUSBX
ch1 ch3 ch5 ch7 ch25 ch27 ch29 ch31
APORT2X
BUSAX
ch0 ch2 ch4 ch6 ch24 ch26 ch28 ch30
APORT1X
INP_MUX
+
-
BUSADC0Y
ch0 ch1 ch2 ch3 ch12 ch13 ch14 ch15
APORT0Y
BUSDY
ch0 ch2 ch4 ch6 ch24 ch26 ch28 ch30
APORT4Y
BUSCY
ch1 ch3 ch5 ch7 ch25 ch27 ch29 ch31
APORT3Y
BUSBY
ch0 ch2 ch4 ch6 ch24 ch26 ch28 ch30
APORT2Y
BUSAY
ch1 ch3 ch5 ch7 ch25 ch27 ch29 ch31
APORT1Y
INN_MUX

Silicon Labs' EFR32BG1 Reference Manual

ADC – Selecting inputs

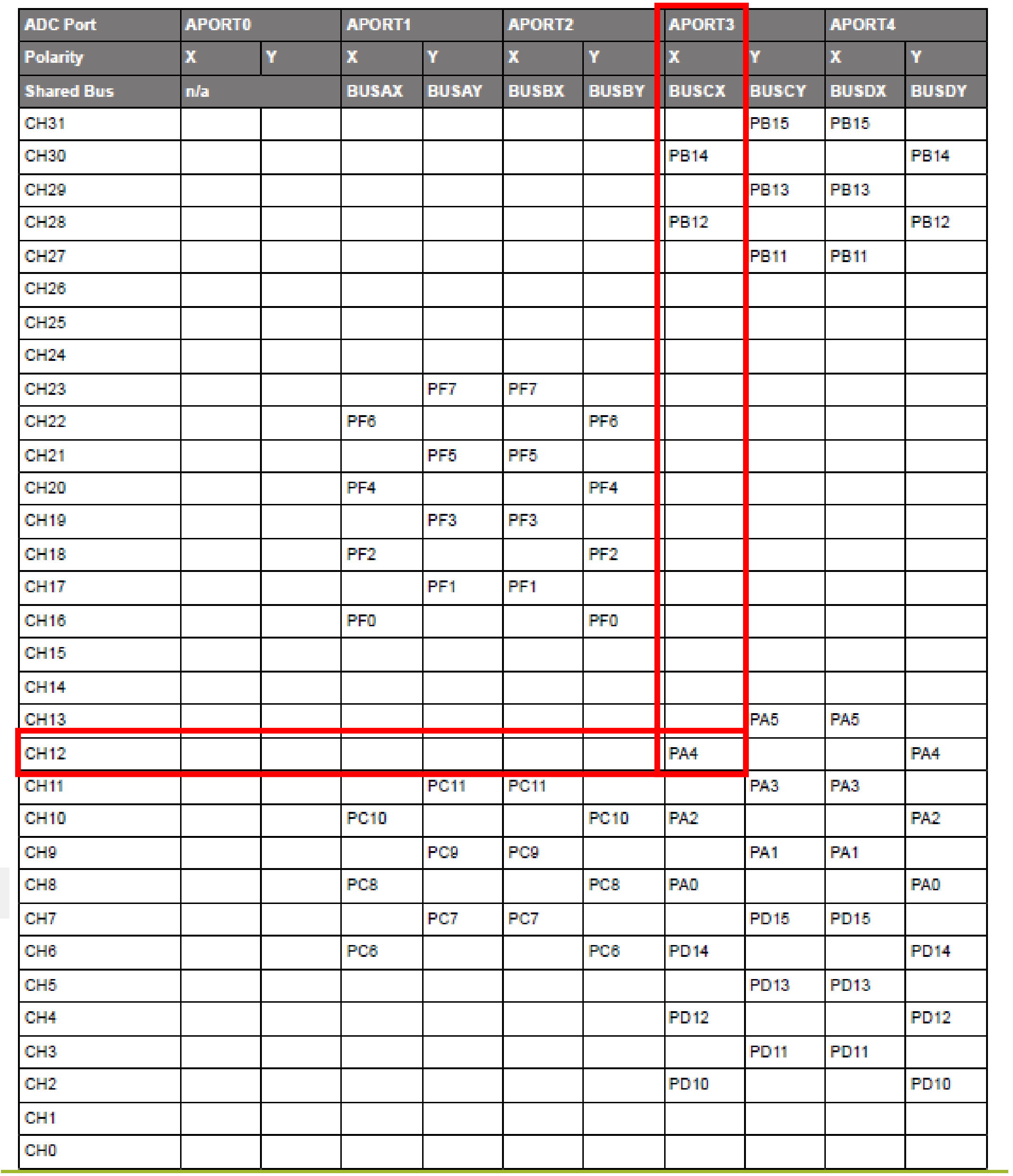
- In the ADC init Type_Def, you can find the proper enumeration to set both the positive and negative inputs to the ADC.
- Ex: Single ended with ADC input on pin PA4
  - adc0_single_init.posSel =



```
    adcPosSelAPORT3XCH12 = _ADC_SINGLECTRL_POSSEL_APORT3XCH12,
```

  - You can find the enumeration using the BG only HAL reference manual

Table 24.1. ADC0 Bus and Pin Mapping

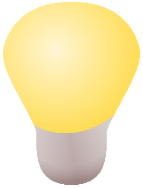


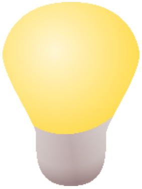
ADC Port	APORT0		APORT1		APORT2		APORT3		APORT4	
Polarity	X	Y	X	Y	X	Y	X	Y	X	Y
Shared Bus	n/a		BUSAX	BUSAY	BUSBX	BUSBY	BUSCX	BUSCY	BUSDX	BUSDY
CH31								PB15	PB15	
CH30							PB14			PB14
CH29								PB13	PB13	
CH28							PB12			PB12
CH27								PB11	PB11	
CH26										
CH25										
CH24										
CH23				PF7	PF7					
CH22			PF6			PF6				
CH21				PF5	PF5					
CH20			PF4			PF4				
CH19				PF3	PF3					
CH18			PF2			PF2				
CH17				PF1	PF1					
CH16			PF0			PF0				
CH15										
CH14										
CH13								PA5	PA5	
CH12							PA4			PA4
CH11				PC11	PC11			PA3	PA3	
CH10			PC10			PC10	PA2			PA2
CH9				PC9	PC9			PA1	PA1	
CH8			PC8			PC8	PA0			PA0
CH7				PC7	PC7			PD15	PD15	
CH6			PC6			PC6	PD14			PD14
CH5								PD13	PD13	
CH4							PD12			PD12
CH3								PD11	PD11	
CH2							PD10			PD10
CH1										
CH0										

Silicon Labs' EFR32BG1 Reference Manual

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();

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 EM3
  - Enable interrupts to the CPU by enabling the ADCn in the Nested Vector Interrupt Control register using NVIC_EnableIRQ(ADCn_IRQn);

Silicon Labs' EFM32 reference manual

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

Firmware Best Practices

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

```
a = (5/2) * 4;            a = 5 * (4/2);            a = (5 * 4) / 2;
a = (2) * 4;              a = 5 * 2;                a = 20 / 2;
a = 8;                    a = 10;                   a = 10;
```

Integer Math

Firmware Best Practices

- Another example
- In “real” math, (5/2) * 3 = 7.5
- In Integer math, ordering matters towards precision

```
a = (5/2) * 3;          a = 5 * (3/2);          a = (5 * 3) / 2;
a = (2) * 3;            a = 5 * 1;              a = 15 / 2;
a = 6;                  a = 5;                  a = 7;
```

If staying strictly in Integer Math, always do the multiplication first. Will limit the loss of accuracy

MCU math 101

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

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

Integer Math

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

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:

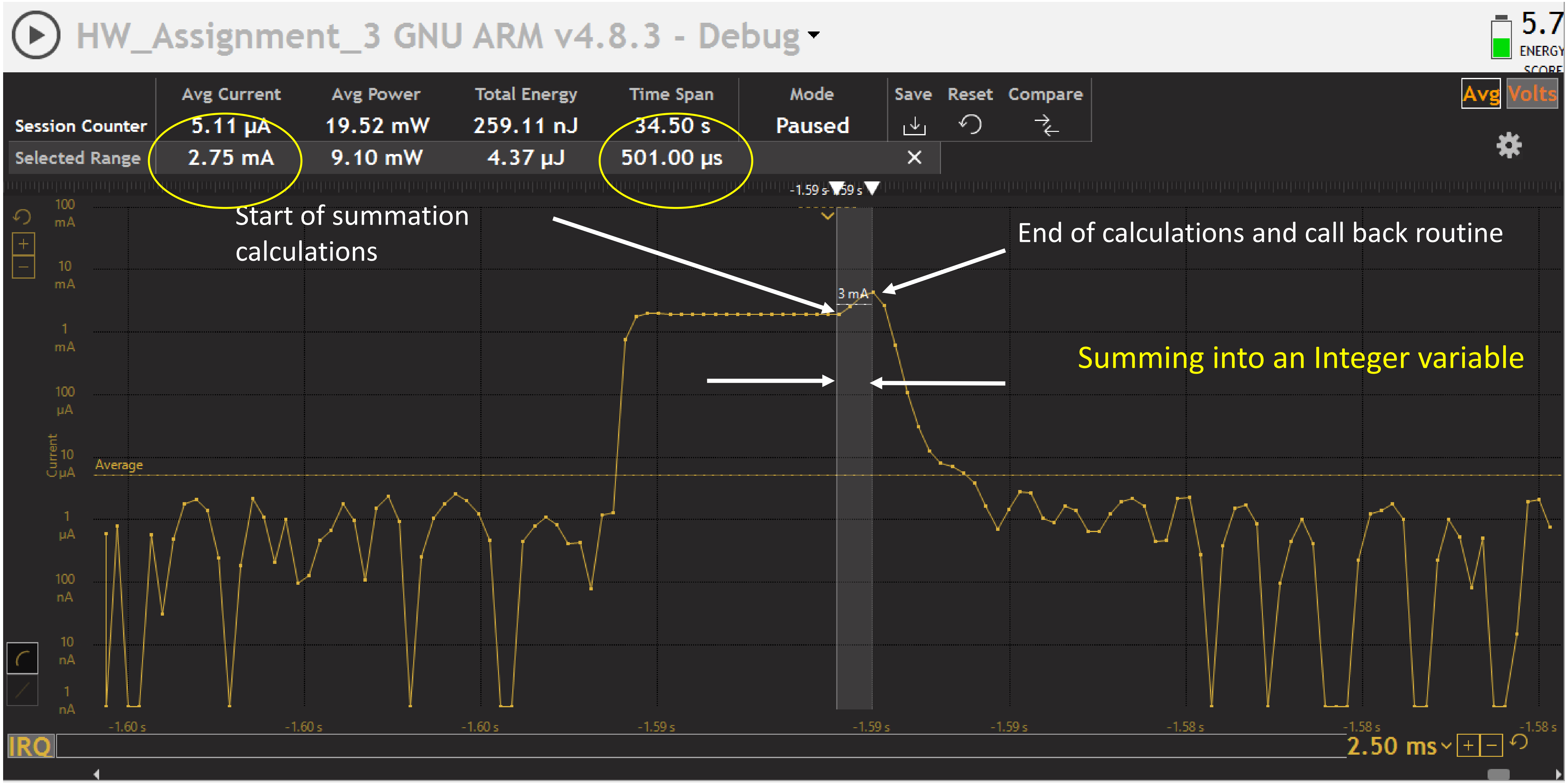
```
    int ramBufferData[BufferSize];
    int Summation;

   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

	Avg Current	Avg Power	Total Energy	Time Span	Mode
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

Integer summation/averaging of 200 ADC samples

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:

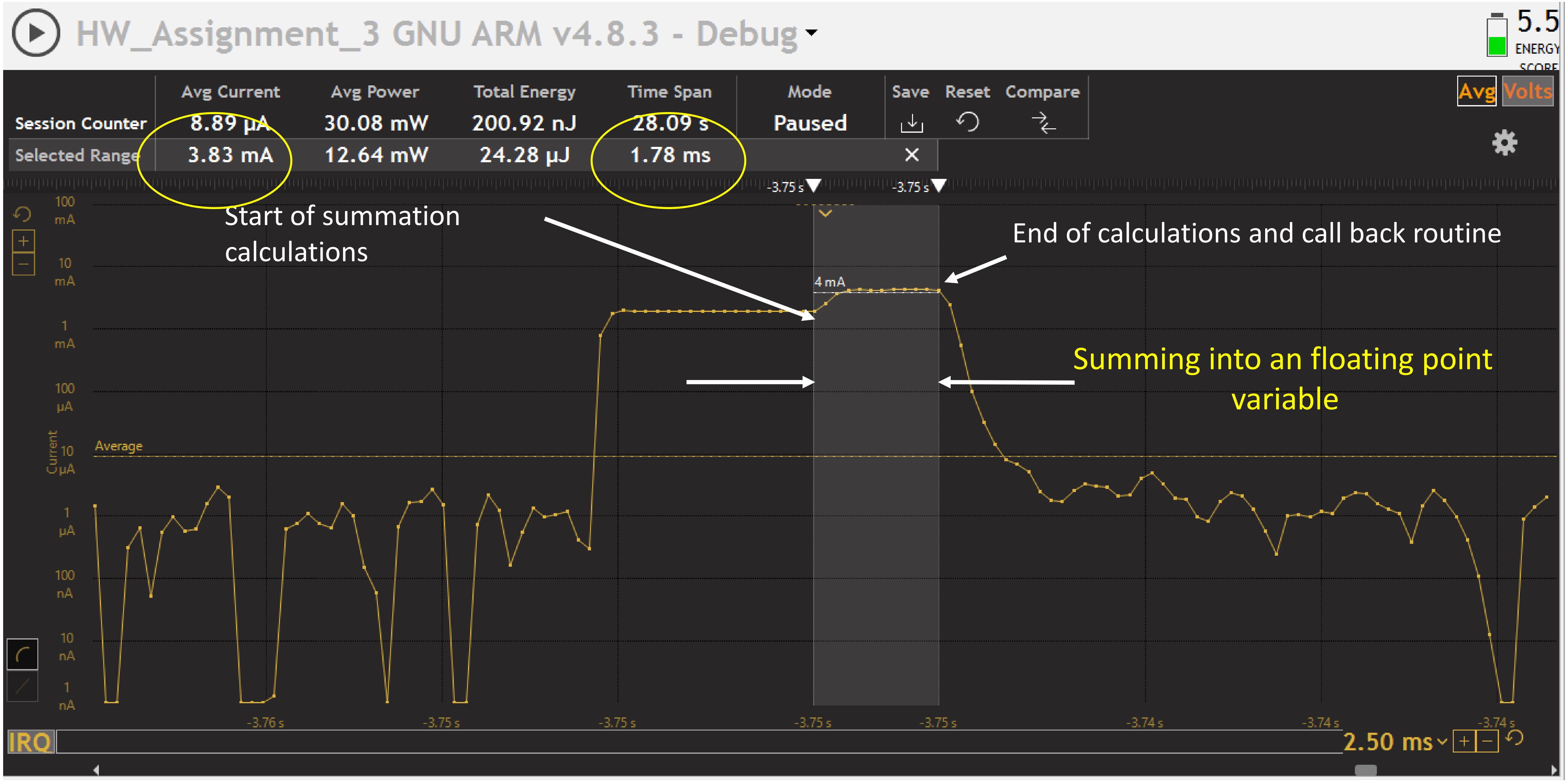
```
Summation = Summation + ramBufferAdcData[j];
00005452:  ldr    r3,[pc,#0x90] ; 0x54e0
00005454:  ldr    r2,[sp,#0x1c]
00005456:  ldrh.w  r3,[r3,r2,lsl #1]
0000545a:  mov    r0,r3
0000545c:  bl     0x00005804
00005460:  mov    r3,r0
00005462:  ldr    r0,[sp,#0x18]
00005464:  mov    r1,r3
00005466:  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
0000581c:  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
```

Disassembled code from Simplicity Studio



HW_Assignment_3 GNU ARM v4.8.3 - Debug

	Avg Current	Avg Power	Total Energy	Time Span	Mode
Session Counter	8.89 µA	30.08 mW	200.92 nJ	28.09 s	Paused
Selected Range	3.83 mA	12.64 mW	24.28 µJ	1.78 ms	

Start of summation calculations

End of calculations and call back routine

Summing into an floating point variable

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

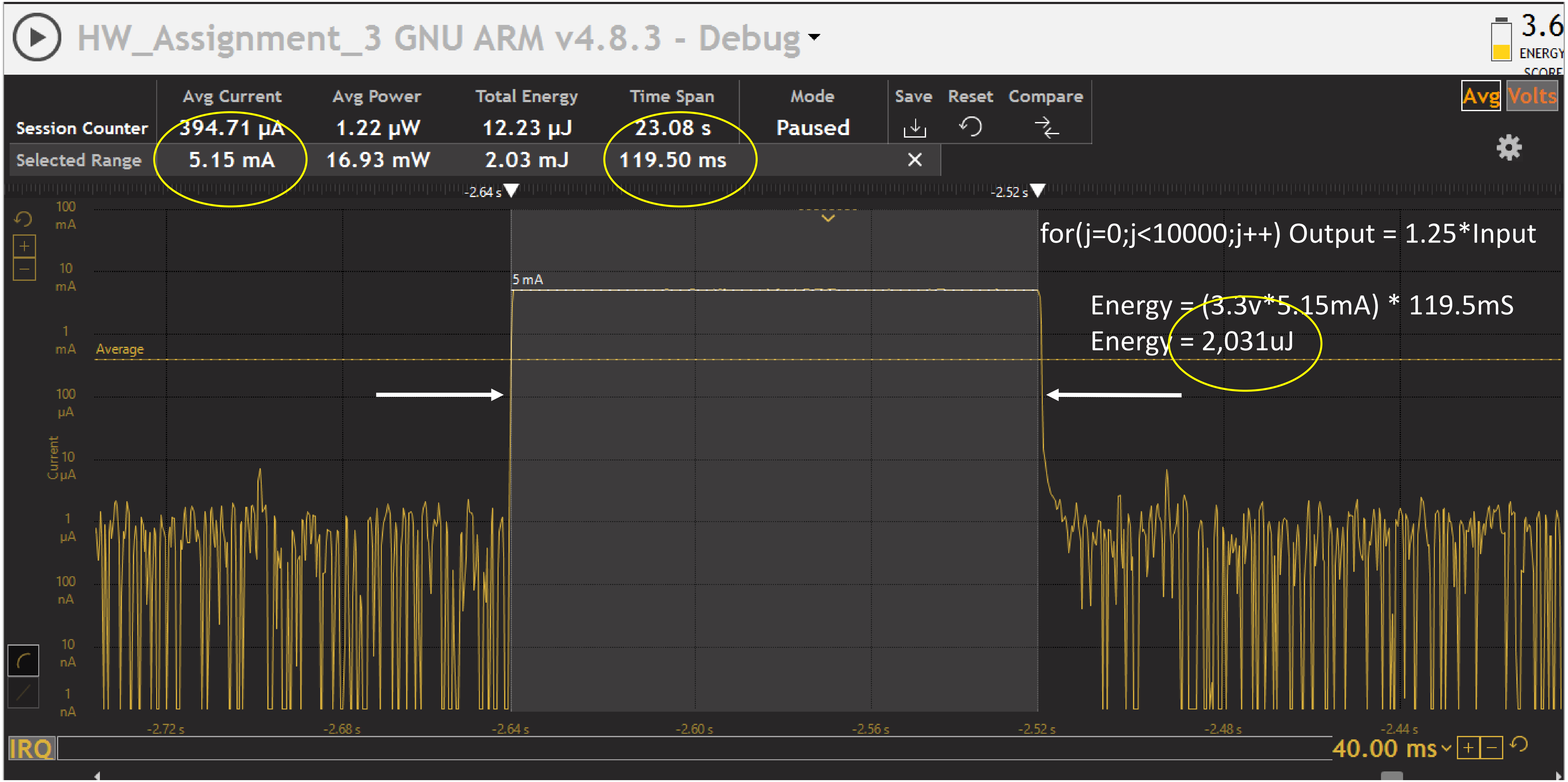
Cortex-M3: Strength in Integers

- 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;
}
```



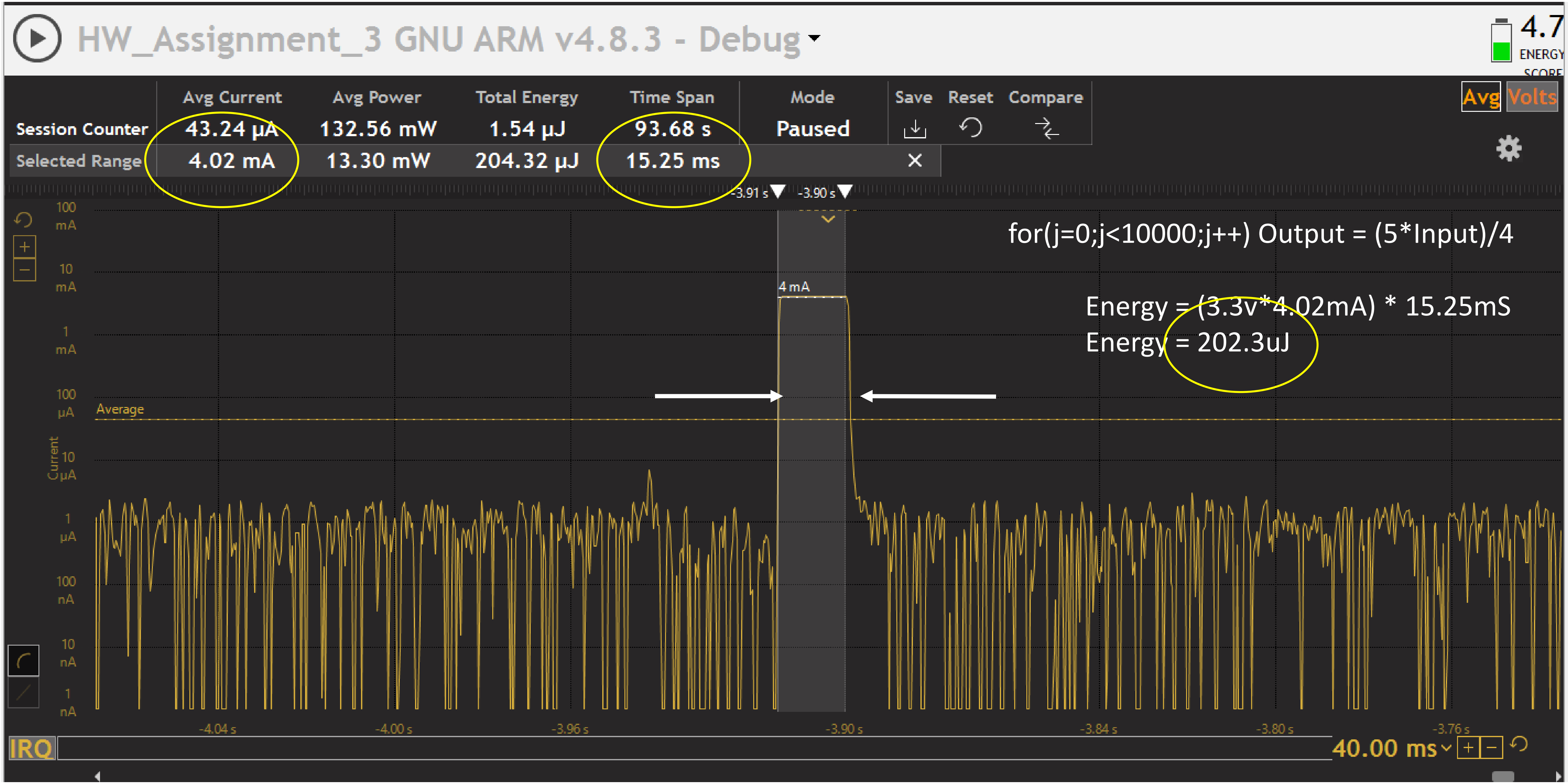
HW_Assignment_3 GNU ARM v4.8.3 - Debug

	Avg Current	Avg Power	Total Energy	Time Span	Mode
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	

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

Energy = (3.3v*5.15mA) * 119.5mS

Energy = 2,031uJ



	Avg Current	Avg Power	Total Energy	Time Span	Mode
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	

Save Reset Compare

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

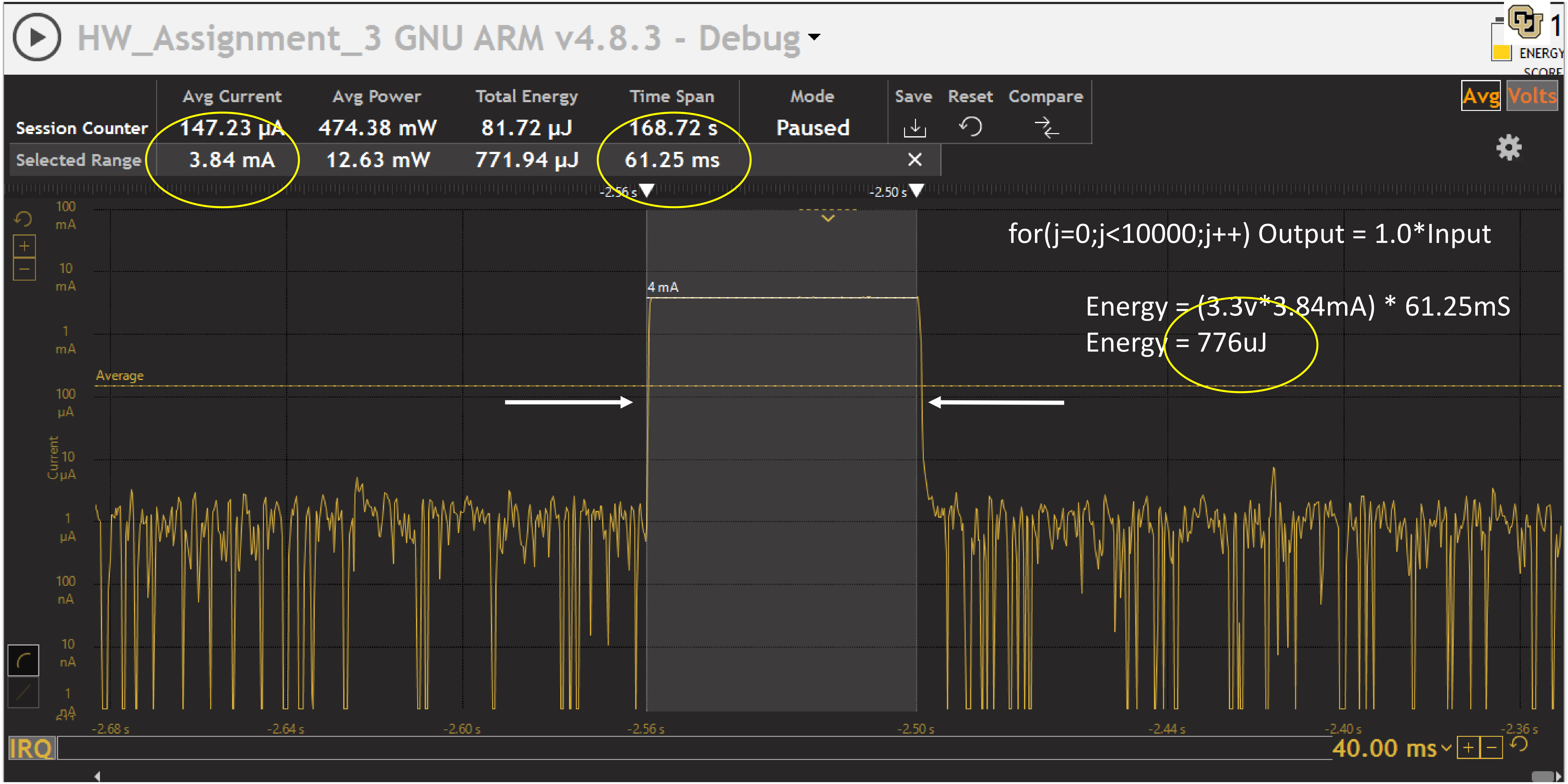
Assembly code for Output = (5*Input)/4

```
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]
```

Simplicity Studio disassembler

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



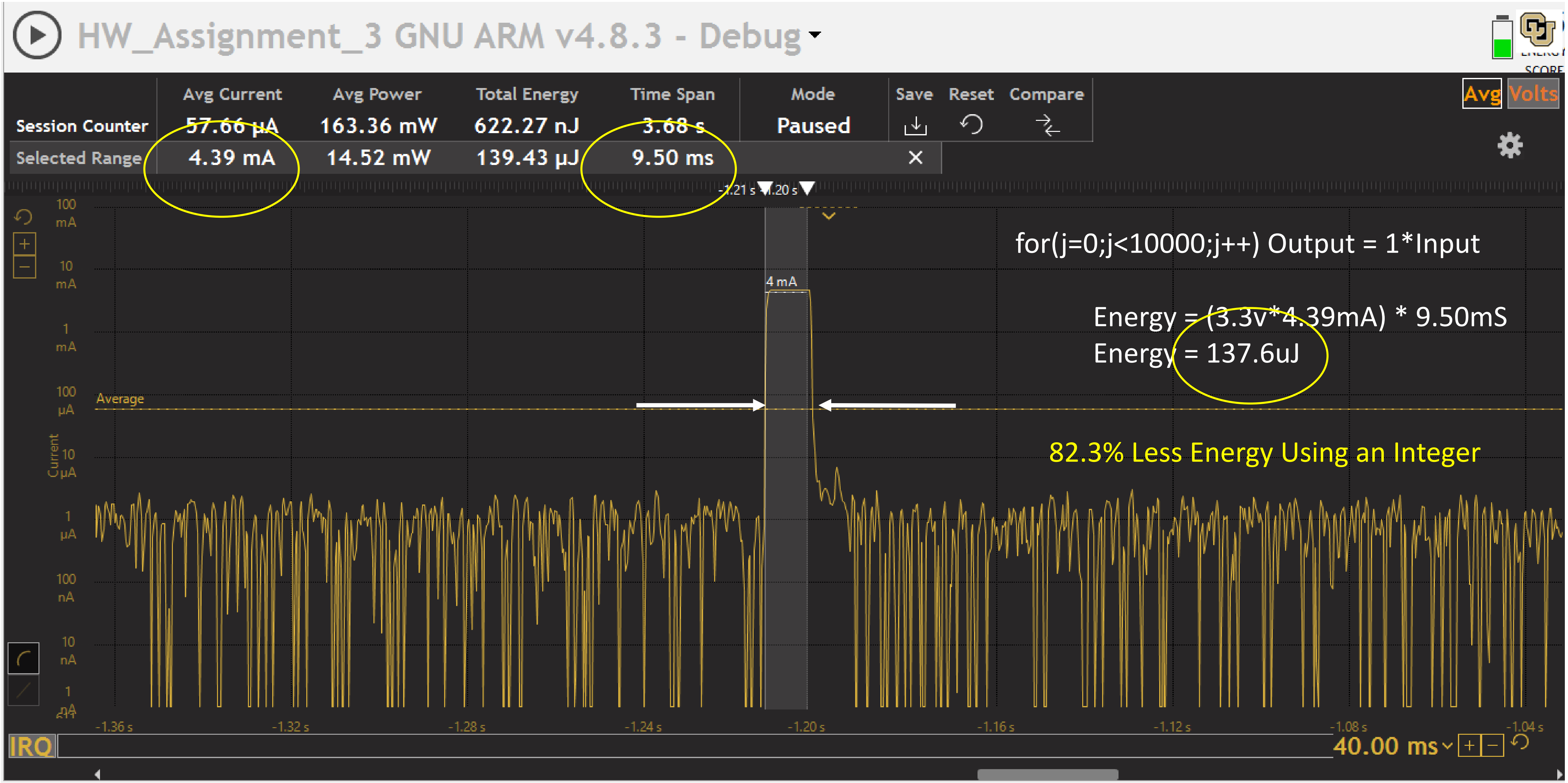
HW_Assignment_3 GNU ARM v4.8.3 - Debug

	Avg Current	Avg Power	Total Energy	Time Span	Mode
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	

for(j=0;j<10000;j++) Output = 1.0*Input

Energy = (3.3v*3.84mA) * 61.25mS

Energy = 776uJ



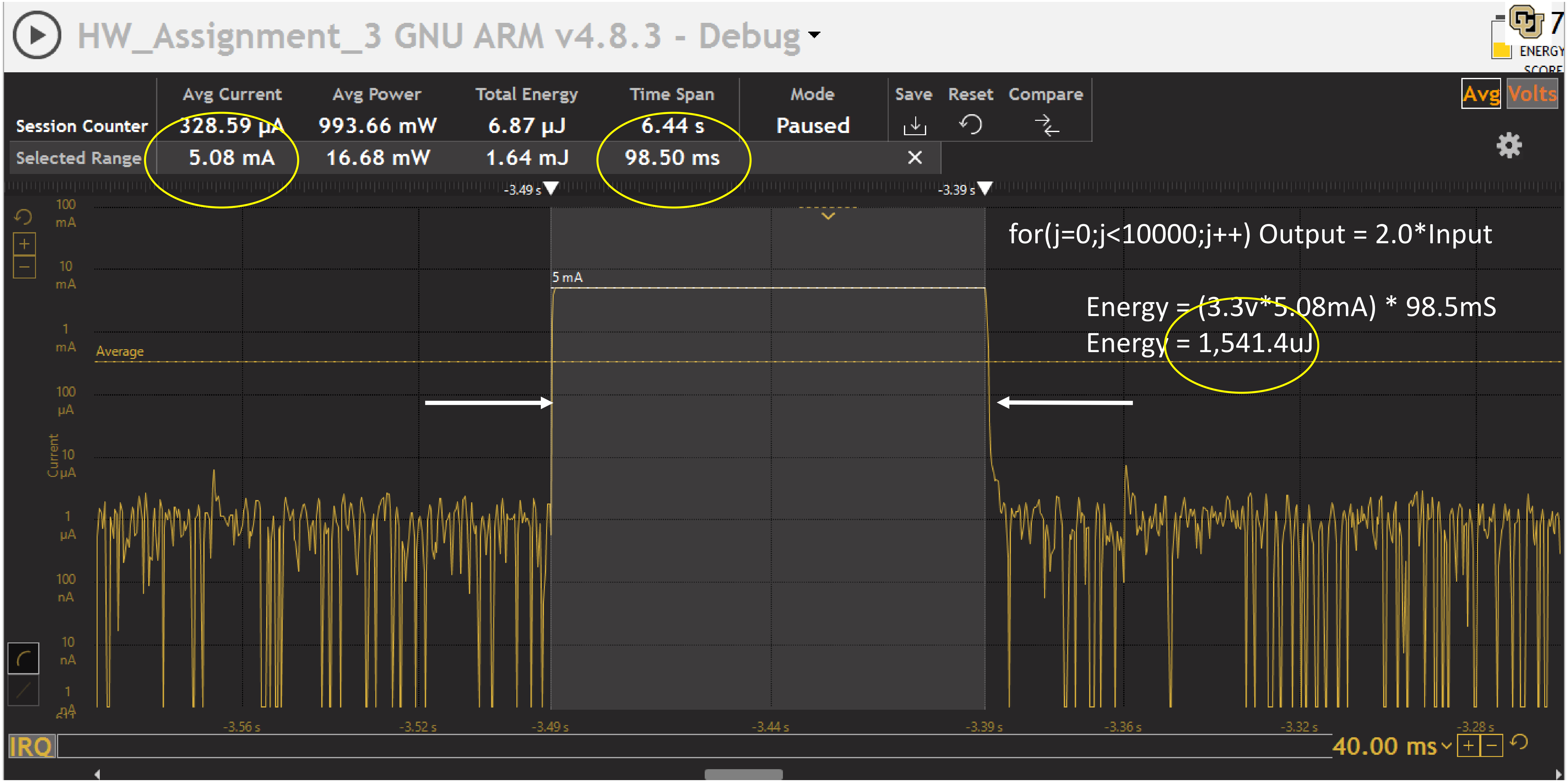
HW_Assignment_3 GNU ARM v4.8.3 - Debug

	Avg Current	Avg Power	Total Energy	Time Span	Mode
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	

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



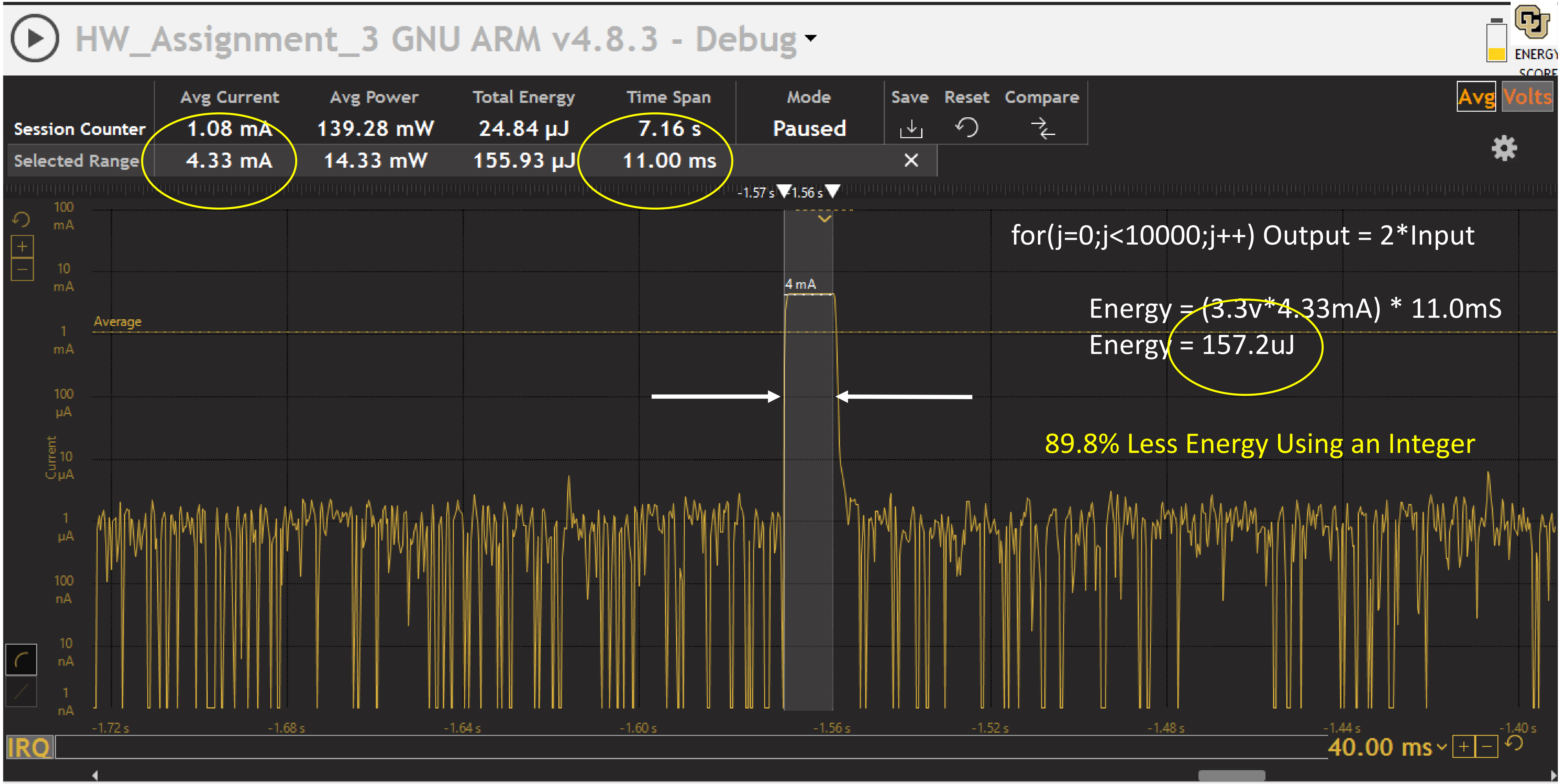
HW_Assignment_3 GNU ARM v4.8.3 - Debug

	Avg Current	Avg Power	Total Energy	Time Span	Mode
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	

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

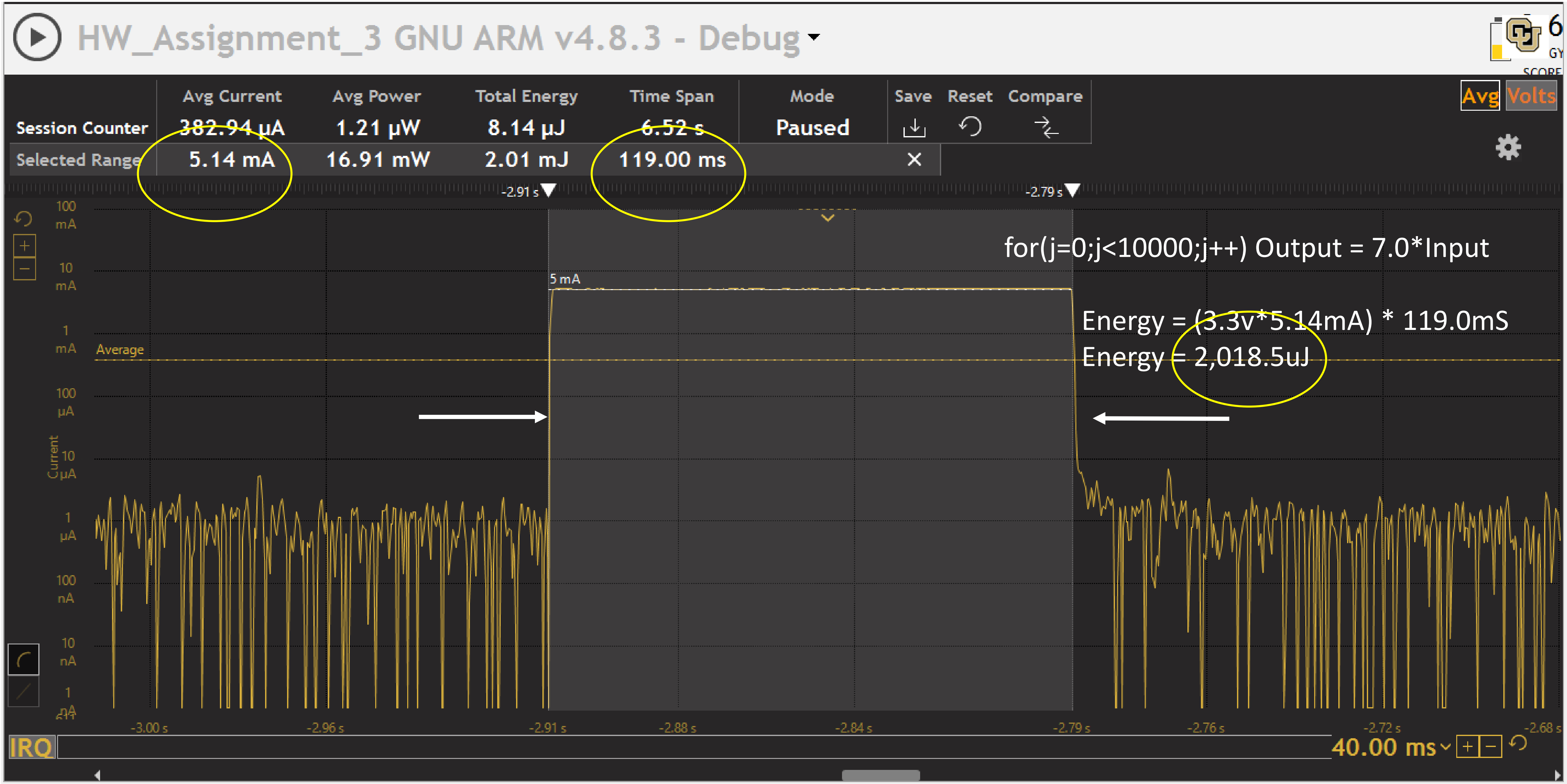
	Avg Current	Avg Power	Total Energy	Time Span	Mode
Session Counter	1.08 mA	139.28 mW	24.84 μJ	7.16 s	Paused
Selected Range	4.33 mA	14.33 mW	155.93 μJ	11.00 ms	

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



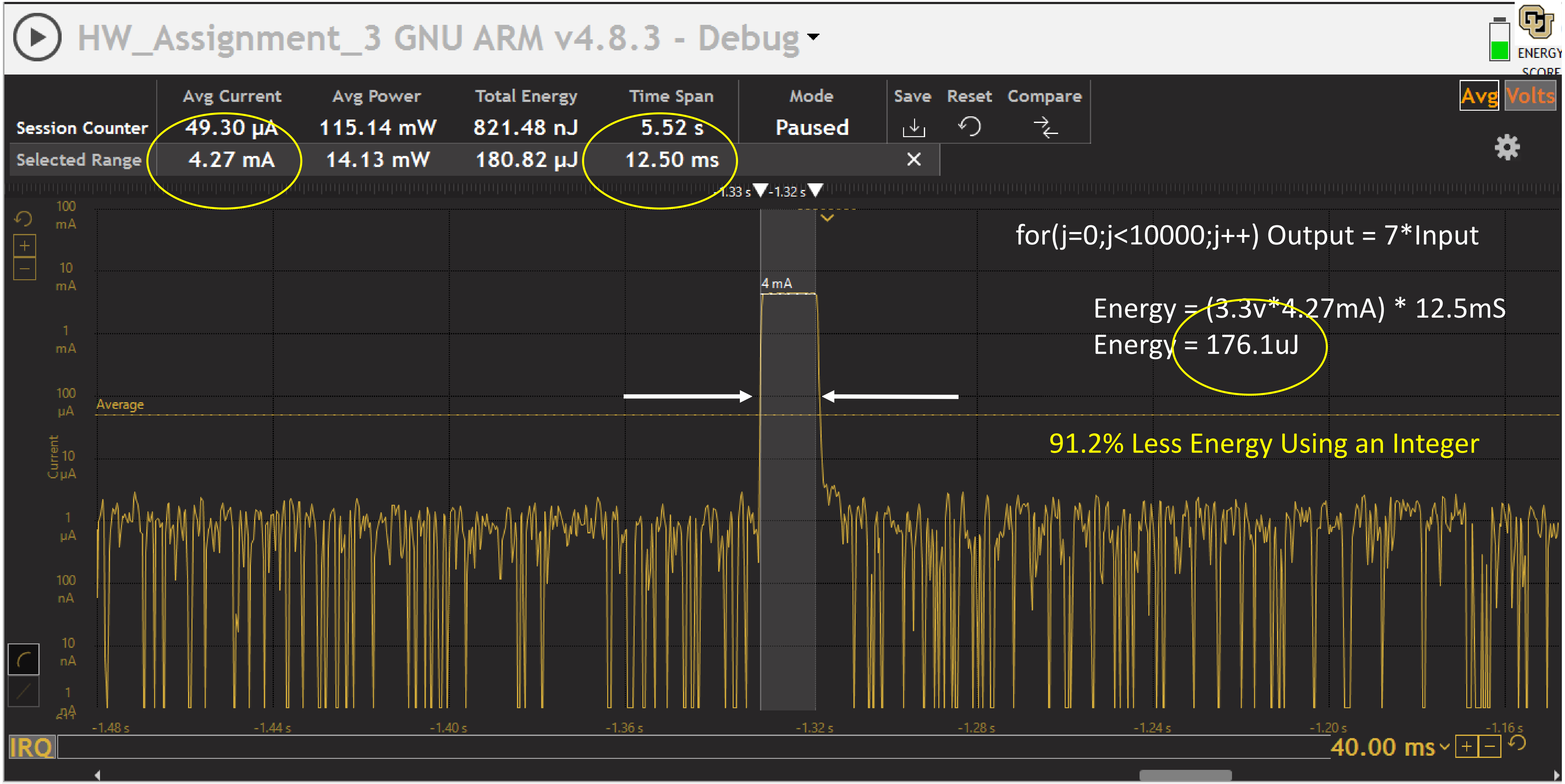
HW_Assignment_3 GNU ARM v4.8.3 - Debug

	Avg Current	Avg Power	Total Energy	Time Span	Mode
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	

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

	Avg Current	Avg Power	Total Energy	Time Span	Mode
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	

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