Hybrid and Adaptive Digital Filter architecture for Robust tracking of On-chip Low Frequency Oscillator Period, enabling Crystal less BLE operation in Low Power Wireless MCUs

Anurag Choudhury– Texas Instruments (India) Pvt. Ltd.

Robin Hoel – Texas Instruments (Norway) Pvt. Ltd.

Torjus Kallerud – Texas Instruments (Norway) Pvt Ltd.

Ashutosh Mishra – Texas Instruments (India) Pvt. Ltd.

62 THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA® IEEE Council on Electronic Design Automation sigda acm

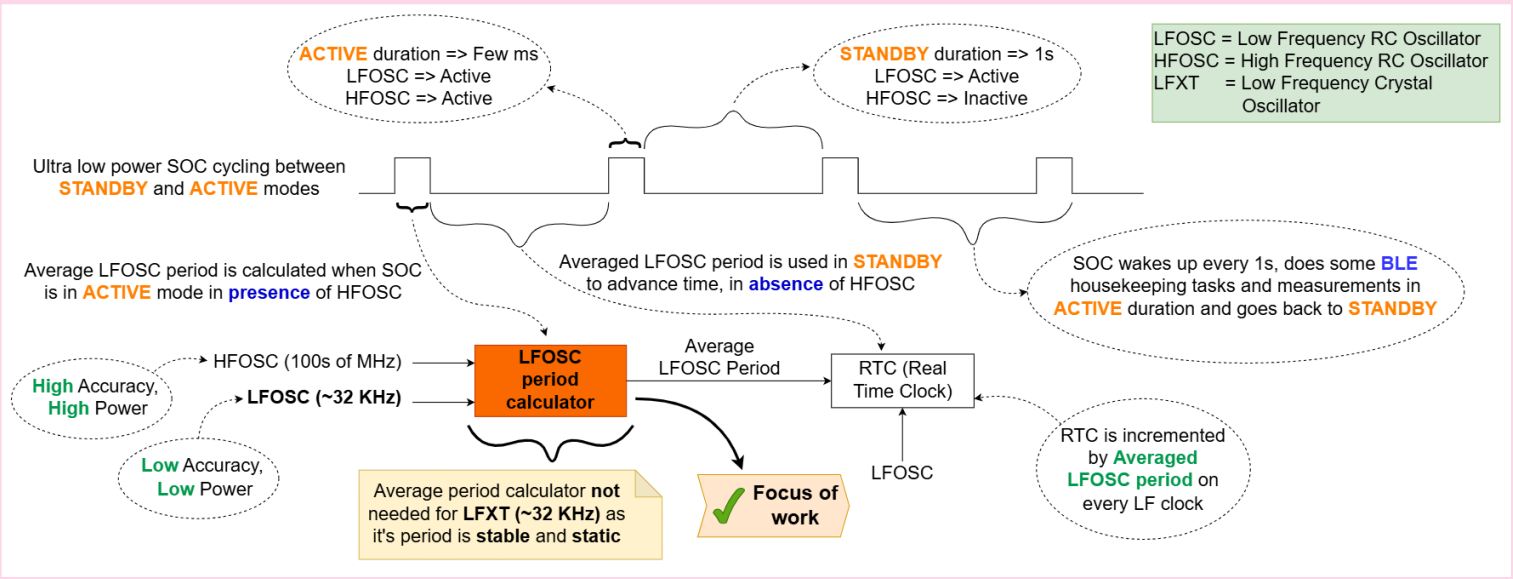
Anurag Choudhury

Design Engineer,
Texas Instruments India Pvt Ltd

TEXAS INSTRUMENTS

Problem Statement / Motivation

- Industry is moving towards **crystal less** SOC designs to save on cost and board area
- A low power BLE MCU spends **>99%** time in **STANDBY** mode (**low** power consumption mode achieved via **power gating**) and **<1%** time in **ACTIVE** mode (**high** power consumption mode)
- Accurate time keeping is difficult in STANDBY mode as the precise high frequency oscillators have to be disabled. The SOC relies on a Low frequency oscillator and it's **pre-calculated** clock period to maintain system time. This is demonstrated in the below figure:-



Focus of this work:
A novel filter design that measures average period of LFOSC (also shown in figure)

Problem Statement / Motivation

- Traditionally the low frequency clock was derived from **LFXT**, which was used to maintain accurate system time. Replacing it with LFOSC has the following **challenges**:

Performance criteria	LFXT	LFOSC
Jitter Noise	<3ns	>20ns
Random Telegraph Noise (RTN)	None	Can be as high as 1000ppm
Dependency on oscillator architecture or technology node	None	Oscillator noise characteristics can change

- What is the **end user impact** of switching to LFOSC?

Switching from LFXT to LFOSC → More noise in clock → Difficulty in calculating average clock period → Loss of time keeping accuracy in STANDBY → Loss of BLE connection

- What are we **achieving** from this work?
  - ✓ Enabling a BLE MCU solution without a crystal (i.e. using LFOSC)
  - ✓ Saving crystal cost and PCB area and cost for TI partners
  - ✓ A scalable solution compatible with any kind of oscillator architecture
  - ✓ A scalable solution which provides power vs accuracy tradeoff for use in non-BLE MCUs
  - ✓ Being an industry wide issue, provides TI a competitive edge in Low Power BLE market

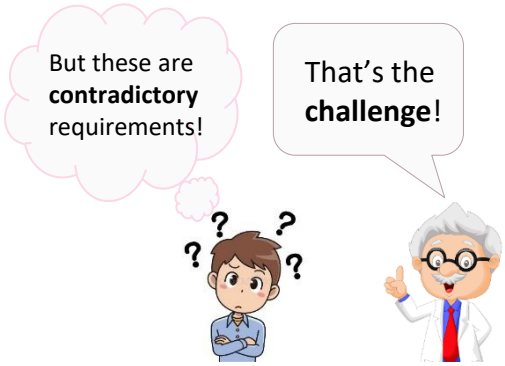
LFOSC Period Estimation Hurdles

Before delving into the solution, let's look at the noise characteristics of LFOSC

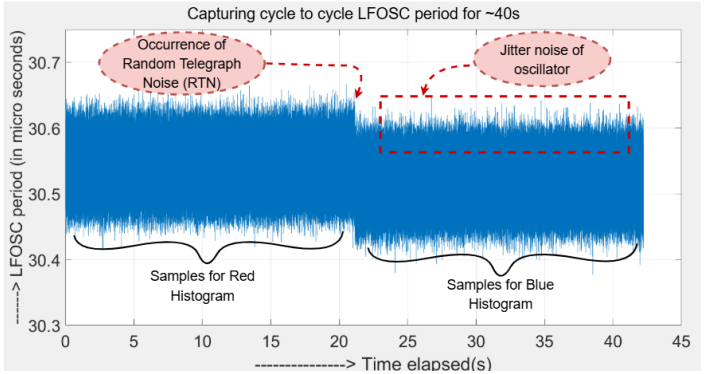
There are **2** sources on noise in LF clock:-

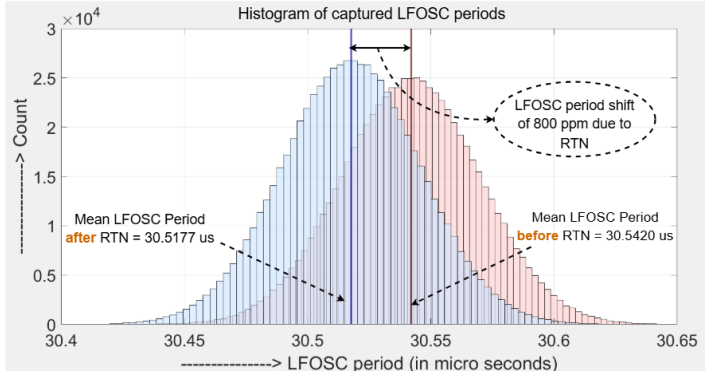
a) Jitter noise
- These are cycle to cycle variations in LFOSC period
- We need a **low bandwidth** Low Pass Filter (LPF) to reject this noise

b) Random Telegraph Noise (RTN)
- These cause a sudden change in mean LFOSC frequency
- We need a **high bandwidth** LPF to settle to the new LFOSC frequency



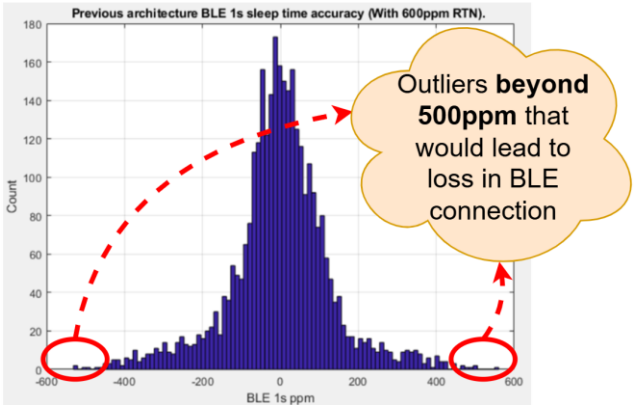
In the below plots, the two types of noise is illustrated:-





Prior Art

- Although a 1st order low pass filter based previous architecture seemed like a good candidate for measuring the average clock period of LFOSC, but it has the following **drawbacks**:
  - X Fast filter response was **too fast** and it captured significant jitter noise while settling to the new LFOSC period
  - X Slow filter response was **too slow** to calculate average LFOSC period with desired accuracy in given time
  - X Lacked a robust mechanism to **detect** LFOSC frequency shift due to RTN **as soon as possible**
  - X Very difficult to **tune** the filter for **varying** oscillator noise characteristics and strike the right balance between accuracy and power consumption
  - X Did not give the statistics backed **confidence** to achieve 1s sleep time accuracy of **500ppm**, which led to failure observations in silicon. This would have led to **loss of BLE connection**.
- The below histogram shows BLE 1s sleep time accuracy (for prior art filter) for **~3600** 1s standby durations, with **600ppm** of RTN injected randomly and **28ns** of cycle to cycle jitter:



What is BLE 1s sleep time accuracy?

- When an SOC goes to sleep for 1s, it's the difference between the **actual sleep time** as compared to elapsed **real world** 1s, once the SOC wakes up:

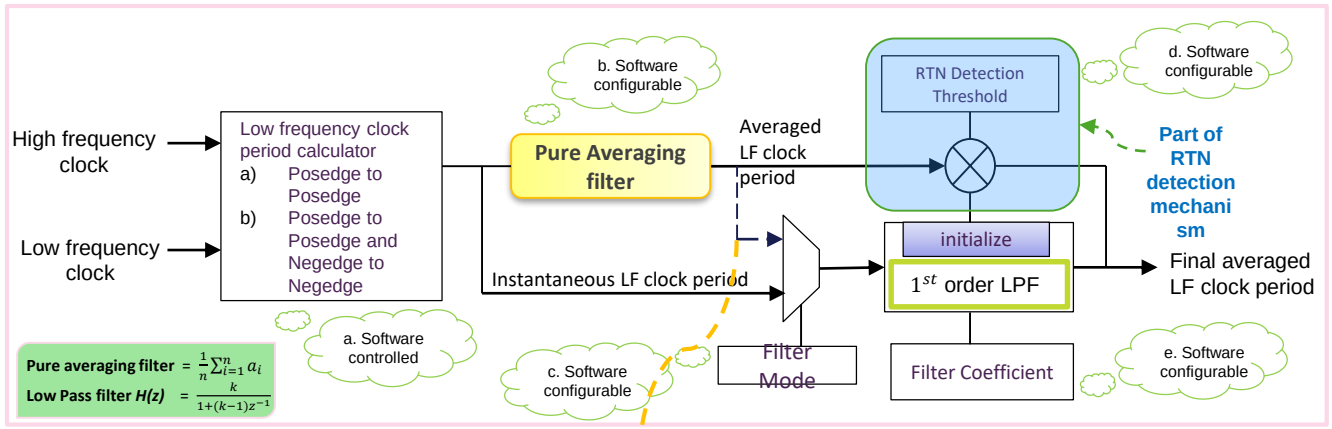
$$\frac{|1 - actual\ sleep\ time|}{1} * 10^6$$

- The max tolerable error in actual wakeup time for 1s sleep window is **500us (or 500ppm)** (as per BLE spec)

Proposed solution

Depending on the selected **filter mode**, the average LFOSC period calculation measurement can run for either **short duration** or **long duration** or **both**. Dynamic selection of measurement duration **saves power**.

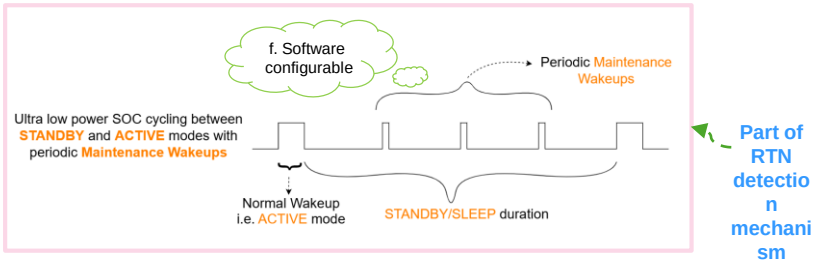
Filter Mode 1 (Dynamic Run time):



Key Highlights

- **Lower power consumption** due to **dynamic** window selection
- **Faster** average period calculation

Disabled path in Filter Mode 1



Algorithm for RTN detection:-

1. Compare the **current** result of 1st order LPF with pure averaging filter at the end of **short duration** window
2. If the error is beyond the RTN threshold error
   - Continue the pure averaging for the remaining duration of **long window**
   - At the end of long window **re-initialize** the LPF with the result from pure averaging circuit
   - **Reset** the pure averaging filter and go back to step 1

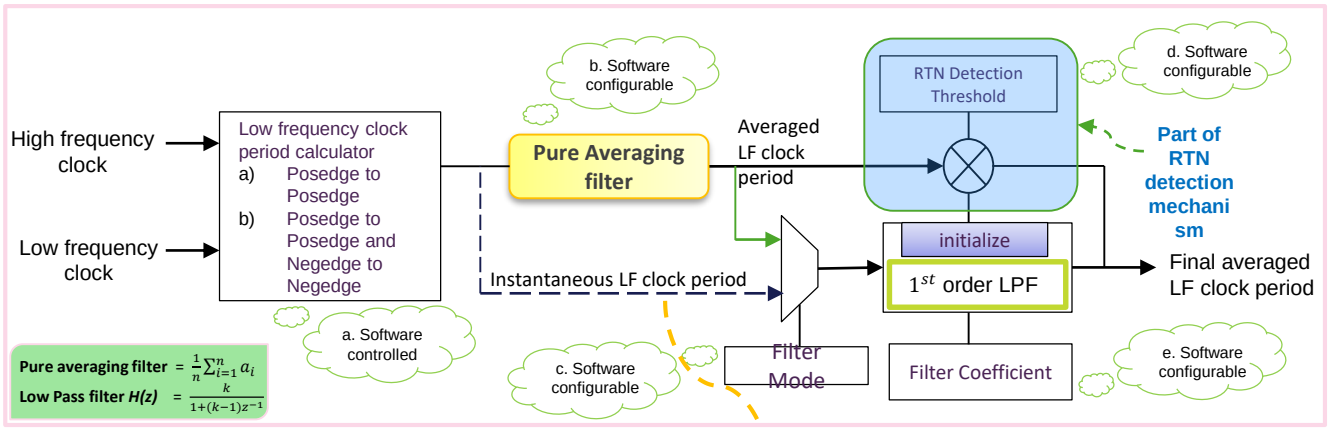
   else
   - **Reset** the pure averaging filter and go back to step 1

General scenario without RTN:-

The 1st order LPF is updated at **every** LF clock edge

Proposed solution

Filter Mode 2 (Static Run time):



Disabled path in Filter Mode 2

Key Highlights:

- **Higher power consumption** due to **static** measurement window size selection
- **Slower** average LFOSC period calculation time as compared to Filter Mode 1

Algorithm for RTN detection:-

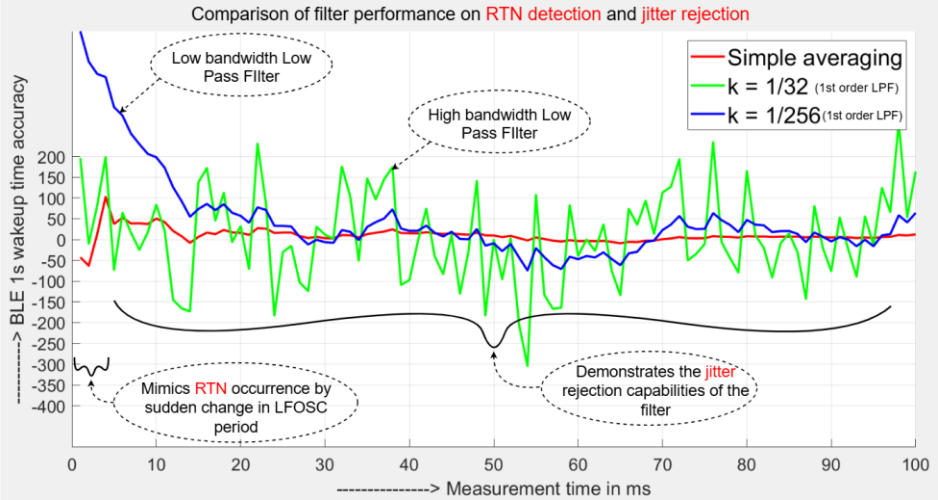
1. Compare the **previous** result of 1st order LPF with pure averaging filter at the end of **long duration** window
2. If the error is beyond the RTN threshold error
   - **Re-initialize** the LPF with the result from pure averaging circuit
   - **Reset** the pure averaging filter and go back to step 1

   else
   - **Reset** the pure averaging filter and go back to step 1

General scenario without RTN:-

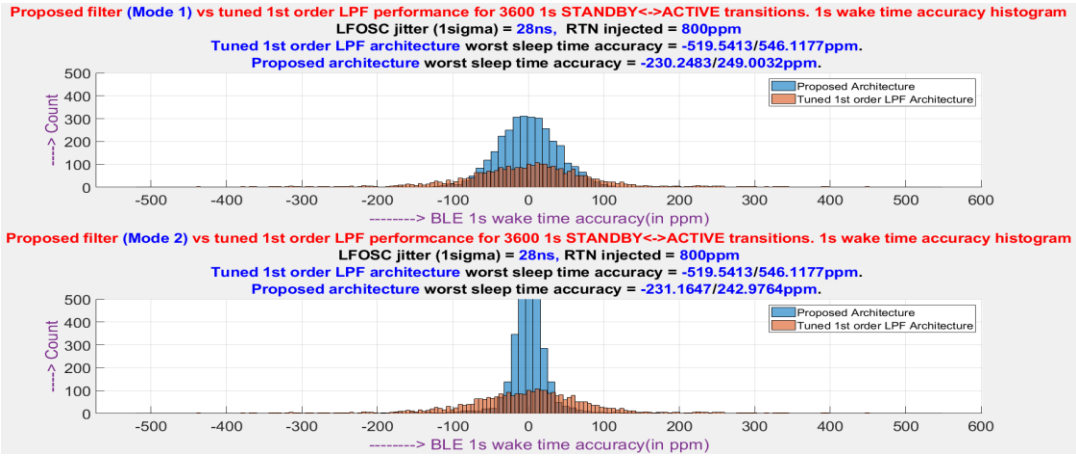
The 1st order LPF is updated at the **end** of **long duration** measurement window

Results



The above plot shows the **1s** wakeup time accuracy in ppm(compared to real world time) for **given** measurement window of filter operation(indicated by x-axis).

Scenario	Parameter	Filter	Value
RTN correction	Settling time	Tuned LPF	~30ms
		Proposed filter	**~<10ms**
Jitter noise rejection	Time drift over 1s	Tuned LPF	~75ppm
		Proposed filter	**~20ppm**



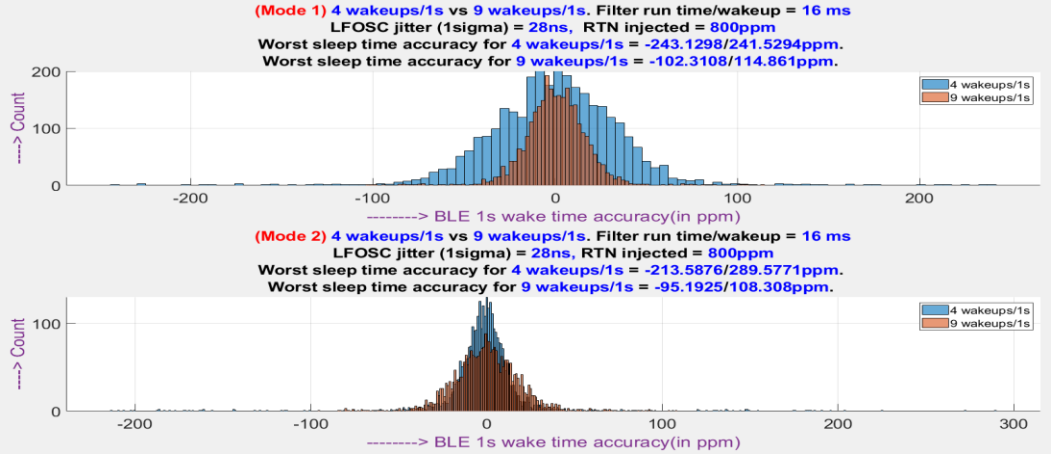
Above plot shows histogram comparison of **1s** wakeup accuracy. There are **4 wakeups** in 1s (i.e. **3** additional **maintenance** wakeups), with window size of **4ms** each. Noise details mentioned in figure.

Tuned LPF vs Proposed Filter Mode 1 & 2 (Filter run time/wakeup = 4ms)

Operation Mode	Parameter	Tuned LPF	Proposed filter
Proposed filter - Mode 1	Worst case outlier (ppm)	546ppm	**249ppm**
	Standard Deviation (ppm)	110ppm	**42ppm**
Proposed filter - Mode 2	Worst case outlier (ppm)	546ppm	**242ppm**
	Standard Deviation (ppm)	110ppm	**32ppm**

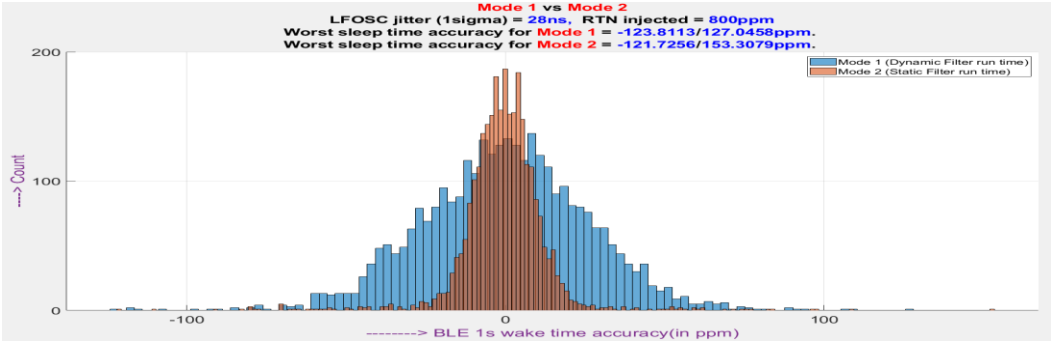
Results

Proposed filter vs Wakeups/1s STANDBY window (Filter run time/wakeup = 16ms)



Parameter	Wakeups/1s window (Maintenance wakeups)	Proposed filter - Mode 1	Proposed filter - Mode 2
Worst case outlier (ppm)	4	243ppm	289ppm
	9	**114ppm**	**108ppm**
Standard Deviation (ppm)	4	41ppm	23ppm
	9	**28ppm**	**18ppm**

Mode1 (Dynamic Filter run time/wakeup) vs Mode 2 (Static Filter run time/wakeup) for low-medium jitter oscillator



In **Filter Mode 1**, filter will run for **4ms** when if there is no RTN. Once RTN is detected, filter will run for **16ms**

In **Filter Mode 2**, filter will always run for **16ms** per wakeup

Parameter	Proposed filter - Mode 1	Proposed filter - Mode 2
Worst case outlier (ppm)	127ppm	153ppm
Standard Deviation (ppm)	26ppm	14ppm
Average filter run time (ms)	**4.04ms**	**16ms**

Both the filter modes have **acceptable** performance but Filter Mode 1 has ~**4x** lower run time

Understanding the results

Scenario		Filter Run time	Periodic wakeups	Impact on power consumption
LFSOC jitter	**LFOSC RTN**			
Low	Low	Low in both modes	Minimal (may be 0)	Low
Low	High	Low with Mode 1 Higher with Mode 2	Multiple	Medium
High	Low	High in both modes	Minimal	Medium
High	High	High in both modes	Multiple	High

Conclusion:

- Periodic wakeup count = f(LFOSC RTN magnitude)
- Filter run time = f(LFOSC jitter magnitude, LFOSC RTN magnitude)

Results Summary

	Old/traditional methodology using LPF	Proposed Filter architecture
Jitter rejection capability	1s wake time error of up to 75ppm	1s wake time error of up to **20 ppm**
RTN correction performance	Corrects RTN in up to 40ms	Corrects RTN in **<10ms**
RTN detection performance	Poor or non existent due to large filter settling time	Can detect RTN with high accuracy in **<5ms**
Number of failure counts to meet 500ppm spec in 3600 (~1 hr) 1s STANDBY<->ACTIVE transitions	~10	**0**
Adjusting to multiple use cases based on power<->wake time accuracy tradeoff, using software configurability	NA	Possible
Risk of violating BLE spec	Possible	**Eliminated**

From the results it was concluded that the proposed filter architecture will **always** meet the **500ppm wakeup time accuracy** with **any On-chip** RC oscillator architecture thus enabling **crystal free** BLE operation

- As compared to a tuned LPF, the proposed filter architecture:
  - is **faster** in **correcting RTN**
  - is **better** in **rejecting jitter noise**
  - has **statistically better** performance, which is attributed to **lower** ppm error standard deviation, which in turn gives **higher** accuracy confidence
- **Different modes** of filter operation targeting specific scenarios
- The Proposed filter has oscillator **architecture agnostic** solution
- Measuring LFOSC period for **both** posedge to posedge and negedge to negedge has proven to **reduce** the filter run time by further **20%** for **same** measurement window
- The below table summarizes the major **software knobs** to detect and correct the **2 major** noise types in silicon:

Performance Parameter	Culprit noise type	Most effective Software configurations
Worst case ppm outlier	RTN magnitude	• Configuring the Number of maintenance wakeups • Configuring right RTN detection threshold
Accuracy confidence without RTN	Jitter noise magnitude	• Selecting right filter mode (i.e. Filter Mode 2 for highest confidence) • Configuring the right filter run time/wakeup