Analysis of Variable Sampling Rate in Simulink-Arduino-Bluetooth System

Introduction

The issue you are experiencing—where a Simulink model running on an Arduino exhibits a consistent, fast sample rate when generating a sine wave, but a variable and slower rate when reading an analog sensor and transmitting data via Bluetooth—is a classic problem in embedded real-time systems. The root cause lies in the fundamental difference between a purely computational task (sine wave generation) and a task that involves hardware interaction and slow I/O (analog read and Bluetooth transmission).

The key finding is that your requested sample time of **0.0001 seconds (100 microseconds)** is physically impossible to maintain when performing an analog-to-digital conversion and transmitting the result over a standard Bluetooth serial link.

Root Cause Analysis

The problem is a result of two primary bottlenecks: the **Analog-to-Digital Conversion (ADC) time** and the **Bluetooth Serial Communication time**.

1. The ADC Conversion Bottleneck

The Sine Wave block is a mathematical function that requires only a few clock cycles to compute. It is a non-blocking, fast operation. In contrast, the **Analog Read block** triggers the Arduino's ADC, which is a hardware process with a fixed execution time.

- **ADC Time:** On common AVR-based Arduino boards (like the Uno or Mega), a single `analogRead()` operation takes approximately **100 microseconds (0.0001 seconds)**.
- **The Conflict:** Your requested sample time is exactly 0.0001 seconds. This means the entire time budget for one loop iteration is consumed by the ADC conversion alone, leaving **zero time** for the Simulink scheduler overhead, the core model logic, and, most critically, the Bluetooth data transmission.

2. The Bluetooth Serial Communication Bottleneck

The data transmission rate is the most significant factor causing the observed slowdown and variability. Your Bluetooth module (e.g., HC-05/06) communicates with the Arduino via a serial port at a specific **baud rate** (e.g., 9600 or 115200 bps).

Baud Rate	Bits per Second	Time to Transmit 1 Byte (approx.)
9600	9,600	1.04 milliseconds (0.00104 s)
115200	115,200	86.8 microseconds (0.0000868 s)

Even at the high rate of 115200 baud, transmitting a single data point (e.g., a 4-byte floating-point number plus overhead bytes for framing and delimiters) takes much longer than your 100-microsecond sample time.

3. The Buffer Overflow and Variable Rate

The inconsistent sample rate you observe (0.0001s for 2s, then 0.003s, 0.005s for longer runs) is a classic symptom of a **Serial Buffer Overflow**.

- **Short Run (2 seconds):** For a short period, the Arduino's internal Serial buffer (typically 64 bytes) and the Bluetooth module's buffer can temporarily store the backlog of data. The Android app reads this data quickly, giving the illusion of a fast, consistent rate.
- **Longer Runs (5 seconds, 10 seconds):** As the run continues, the data is generated much faster than it can be transmitted. The buffers fill up, and the Simulink execution loop is forced to **block** (wait) for the Bluetooth link to clear the buffer before it can execute the next sample. This blocking dramatically increases the actual time between samples, leading to the observed slower, variable rates (0.003s, 0.005s). The variability is due to the non-deterministic nature of the Bluetooth link and the Android app's reading speed.

Solutions and Recommendations

To achieve a stable and reliable sampling rate, you must ensure that the sample time is greater than the total time required for the ADC conversion and the data transmission.

Recommendation 1: Increase the Simulink Sample Time

Set a realistic sample time in your Simulink model that is significantly longer than the total time for one loop iteration.

- **Calculation:** If you are using 115200 baud and sending 10 bytes of data (e.g., a 4-byte float, plus delimiters), the transmission time is about 868 µs. Add the ADC time (100 µs) and some overhead (e.g., 50 µs).

- **Action:** Set your Simulink sample time to at least **0.001 seconds (1 millisecond)**. This provides a stable rate of 1000 samples per second, which is a high and achievable rate for this setup.

Recommendation 2: Maximize the Baud Rate

Ensure you are using the highest possible baud rate supported by your Bluetooth module and the Arduino's serial hardware.

- **Action:** Configure your Bluetooth module and the corresponding Simulink Serial Transmit block to use **115200 baud**. Avoid lower rates like 9600 baud, which would limit your maximum sample rate to about 100 samples per second.

Recommendation 3: Optimize Data Transmission

Reduce the amount of data being sent per sample to minimize the transmission time.

- **Use Integers:** Instead of sending a floating-point number (4 bytes), send the raw 10-bit ADC value as a 2-byte integer (`uint16`). This halves the data size and transmission time.
- **Data Packing:** If you are sending multiple sensor readings, pack them into a single, compact binary packet instead of sending them as separate, delimited strings. This reduces the overhead bytes.

By implementing these changes, particularly setting a realistic sample time (Recommendation 1), you will eliminate the buffer overflow and achieve the consistent, predictable sampling rate required for your application.