Synchronization Of Wireless Accelerometer Sensors For Industrial Application

Swarna Narayanan

Synchronization Of Wireless Accelerometer Sensors For Industrial Application

by

Swarna Narayanan

in partial fulfillment of the requirements for the degree of

Master of Science
in Embedded Systems

at the Delft University of Technology,
to be defended publicly on Monday August 26, 2019 at 01:00 PM.

Student number:	4719905	
Supervisor:	Dr. MSc. Stoyan Nihtianov,	TU Delft, ASML B.V
Thesis committee:	Dr. MSc. Stoyan Nihtianov,	TU Delft, ASML B.V
	Dr. MSc. Qinwen Fan ,	TU Delft
	Prof. Dr. Earl McCune,	TU Delft

Acknowledgements

During this thesis I embarked on a steep learning curve. I would like to express my gratitude to my supervisor Dr. Stoyan Nihtianov for giving me an opportunity to work on an interesting project at ASML and for his invaluable guidance through out the thesis. I also thank my colleagues Mr. Alvaro Torres Di Zeo and Mr. Reza Taherkhani for their insightful discussions and constructive feedback. Mr. Flaviano Tateo from ASML has been very helpful in the testing process. My friends from Eindhoven and Delft have been a pillar of strength and a constant motivation all along this journey. I am grateful to my parents and sister for believing in me and always being there.

நன்றி

(Thank You)

Swarna Narayanan
Eindhoven, August 2019

Abstract

Wireless sensor networks play a vital role in major technological developments. The success of such networks depend on the quality and reliability of data acquisition. Despite a lot of research involving network clock synchronization, the area of synchronous sampling has not been dealt in much detail. This thesis aims at studying and implementing synchronization at the sensor level on wireless accelerometer sensors. The designated application for this thesis involves machine condition monitoring using accelerometer sensors that requires high synchronization accuracy between the samples. Two approaches are presented to obtain synchronous sampling, namely: real-time and subsequent synchronization. The sensor nodes designed with commercially-off-the-shelf components are used for the implementation of the two methods. The embedded software was developed as a platform to realize the proposed synchronous data acquisition techniques. The real-time synchronization uses a software solution and provides an interrupt-based triggering to align the sampling instant of the sensors. An external reliable clock source is required for the accelerometer to implement this technique. The signal propagation delay were minimized by employing interrupts whenever possible. In subsequent synchronization, the samples are collected asynchronously by the sensor nodes. Each sample from the sensor is assigned a timestamp according to its local clock. As a post-process, the time shift between the samples collected by the sensor nodes are estimated and realigned using cross-correlation, interpolation and re-sampling. The subsequent synchronization technique can be used when real-time network synchronization is not possible. The testing environment was designed to emulate the real application. A mechanical shaker was used to provide controlled input signals to the sensors in order to synchronously reconstruct the signals in time-domain.

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

ADC Analog-to-digital converter. 7, 11, 12, 33

API Application programming interface. 19

BLE Bluetooth low energy. 8, 33

EEPROM Electrically erasable programmable random access memory. 7

FRAM Ferroelectric random access memory. 7, 8

I^{2}C Inter-integrated circuit. 7

MCU Microcontroller unit. 9, 23, 25, 27

MEMS Microelectromechanical systems. ix, 9–11, 31

ODR Output data rate. 12, 13

PCB Printed circuit board. 7, 8

ppm Parts per million. 5, 8, 25, 36

PWM Pulse width modulation. 24

RMS Root mean square. 13

SPI Serial peripheral interface. 7, 12

WSN Wireless sensor network. 1, 3, 5

1 Introduction

A Wireless sensor network (WSN) is a collection of spatially distributed sensors that gather information at different locations and transmits to the sink through wireless links. The applications of such networks are vast and ever-growing, the major advantage being the ease with which they can be deployed and removed whenever required. However, the greatest challenge lies in combining the measurements and processing them from different sensor nodes to obtain a better interpretation of the environment. Achieving this, requires the knowledge of precise measurement instances at all locations, which calls for synchronization at the data acquisition phase. Despite a lot of research involving network clock synchronization, the area of synchronous sampling has not been dealt in much detail.

This master's thesis aims at studying, implementing and validating synchronization at the sensor level on wireless accelerometer sensors. An accelerometer is an inertial sensor that measures the vibrations of the platform on which it is placed. These accelerometer sensor nodes have been proposed to diagnose the dynamics of wafer handlers located in the photo-lithographic machines of ASML. The diagnosis involves performing experimental modal analysis of the data obtained. However, in order to extract reliable modal information, the samples must be synchronized with very low tolerance for error. Furthermore, a sufficiently high sampling rate is required to capture high frequency components.

1.1. Objectives

- To study the accelerometer sensor and present a method to achieve sensor-level synchronization.
- To develop embedded firmware as a platform to implement data acquisition and synchronous sampling.
- To align the sampling instances of each sensor within a tolerant limit of 50 μs.

1.2. Contribution

- Embedded software to integrate various peripherals were written. A hardware abstraction layer (HAL) were implemented using device drivers.
- Two approaches to synchronous sampling, namely the real-time and subsequent synchronization techniques were proposed.
- The synchronization approaches were implemented and their accuracy were tested on the accelerometer sensor nodes.

1.3. Thesis Organisation

The thesis is organized into five chapters and is described as follows. Chapter 2 describes the system with respect to its application and requirements. It also provides an account of the hardware system under consideration. The operating principle and working of the accelerometer sensor is explained in detail. A review of current research topics in line with this thesis is also presented. Chapter 3 discusses the embedded platform on which the synchronization approach is implemented. Important aspects of the firmware is detailed. Synchronous sampling is elucidated along with the approach for implementation. Chapter 4 illustrates the testing procedure followed. The results obtained are analysed and presented. Chapter 5 concludes the thesis with recommendations for future work.

2
System Description

2.1. Overview

This chapter gives the necessary background information and related works regarding the project. The application for which the project is to be developed is discussed briefly along with the requirements in Section 2.2. Section 2.3 presents a review of current research work in line with this thesis. The chapter also summarizes the "state of the art" of wireless network and the sensor node used in Sections 2.4 and 2.6.2. The network synchronization for this WSN is summed up in Section 2.5.

2.2. Application and Requirements

The photo-lithography machines of ASML consist of several components and modules, of which the wafer handler comprises the entry and exit stage. A wafer is a thin disc sliced from a silicon cylinder on which, circuit patterns are imprinted. Robots inside the wafer handler transports the wafer for exposing it to photo-lithographic imprinting and return the exposed wafers [1]. The issue with this stage is that these robots often halt suddenly or move irrationally with high acceleration. Therefore, these robots must be monitored to understand its system dynamics. This monitoring is required only at times when diagnosis is necessary and involves acquiring signals from various points on the robots synchronously. Hence, a standalone system that can be deployed when diagnosis is necessary is preferred. Moreover, the wafer handler is already densely packed with its own components which does not leave enough room to install cabled or built-in sensors. Therefore, a WSN standalone system is favored, since it reduces the complexity and hassle of stopping the machine to lay cables or add built-in sensors. The proposed WSN on the wafer handler is illustrated in Figure 2.1.

The wafer flow trajectory in the wafer handler stage is indicated with arrows. The pre-alignment unit centers the wafer before the load robot fetches it and gives it to the next stage. The unload robot transfers the wafer from the previous stage and gives it to discharge unit. The wafer then exits the stage from here. The green cubes on the robots represent the sensor nodes in potential locations for collecting acceleration signals. The sink node is placed near the wafer handler and sends periodic signals to the sensors to keep the clocks synchronized. The sink node receives the required acceleration data and transfers it to the host PC for further off-line analysis. This architecture is proposed for the atmospheric wafer handler. With suitable housing and improved design, the sensor network can also be extended to the vacuum stage.

The approach to diagnosing the wafer handler is to collect signals from accelerometer sensors placed at different locations on the robot. The required sampling rate at all sensor locations is 1000 samples per second. The sampling instants at each location must be synchronized to a maximum

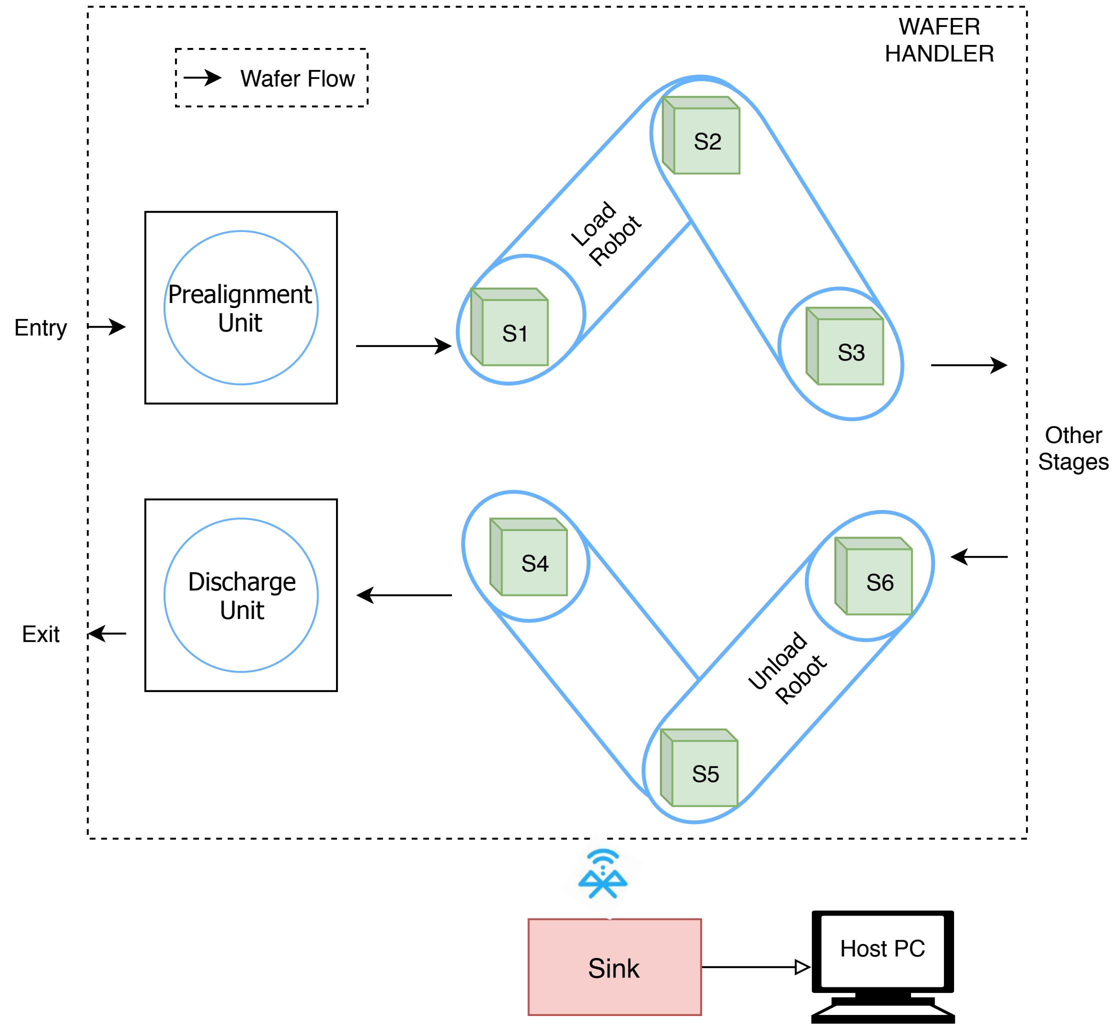


Figure 2.1: Illustration of sensor network on the wafer handler

allowable tolerance of 55.55 micro seconds. This requirement is illustrated in figure 2.2. If t_{11} and t_{21} are the sampling times of the two sensors at to different locations, the difference between them should be less than 55.55 μs.

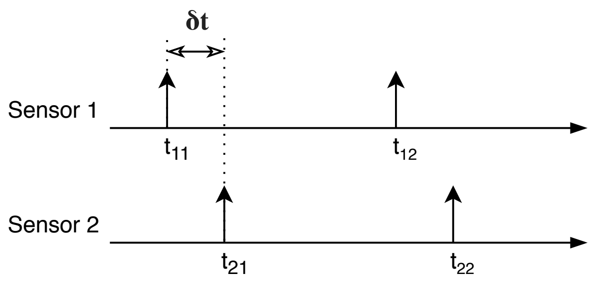


Figure 2.2: Sampling time for two sensors at different locations

The events required to initiate retrieval of the acceleration data collected for 30 seconds from sensors are specified by the stakeholders at ASML. The first event occurs when the machine stops suddenly in the middle of its operation. In this case, the data of interest will be the samples collected shortly before this event was registered. The second event is when the robot's movements exceed an acceleration threshold. For this scenario, the samples collected directly after the event registration are taken into consideration. At a sampling frequency of 1KHz, the number of samples collected is 30,000 for a period of 30 seconds. The time required for the synchronously collected samples to reach the host system is not time-critical. Since the wireless sensor nodes is required to be placed at different locations on the dense wafer handler, it should be small, lightweight and dissipate less heat while consuming power from a battery. These requirements were taken into account when the sensor node was designed, which is elaborated on Section 2.4. The collected data enables to perform experimental modal analysis to understand the complex dynamics of the wafer handler robots. The strict synchronization constraint on the sampling time is specified to avoid any distortion in the resulting mode shapes.

2.3. Related Work

Modal analysis is a popular method for diagnosing the dynamic performance of vehicles, machines and structures in response to external excitation [2]. Natural frequencies, damping factors and vibration mode shapes are the key modal parameters that are to be extracted from the system for further analysis. WSNs can be used to carry out vibration monitoring as they enable effective analysis after data acquisition. In order to reconstruct mode shapes in modal analysis, a frequency domain decomposition (FDD) technique is used. This method measures the variation in shape of the mode based on the synchronicity of the sensor. If considerable error occurs in the time data at a single location, a relative time shift is introduced to the data sampled at that point. This time shift when translated into the frequency domain for modal analysis leads to an error in the resolution of the mode shape [3]. The cited work shows the impact of time synchronization on mode shape reconstruction. Hence, synchronization accuracy in the range of micro-seconds is required to achieve the desirable results for the application.

Time-synchronization in a WSN is a matter of interest in time-critical applications. In applications that involve multi-sensor fusion, exact time stamping of data and clock synchronization is inevitable. This allows for a meaningful understanding of the system. If the timestamps are incorrect, then the estimation of the system behavior cannot be evaluated properly. The latencies introduced into the system must be measured and compensated. The tolerance is usually in the range of a few seconds in environmental monitoring and drops to tens of microseconds in more demanding vibration-based monitoring [4]. For an application with a high sampling rate, even a small synchronization error between sampling points causes major phase distortions [5]. A sensor network consists of several nodes that transmit data to a master sink node. In a distributed sensor network, each node has its own internal clock. The frequency stability of the oscillator varies from sensor to sensor which introduces clock skew relative to each other [6]. The skew is affected by environmental factors such as temperature, humidity and aging. The clock skew is greater than 1, if the physical clock is faster than the real clock. Similarly, the clock skew is less than 1, if the physical clock is slower than the real clock. This is represented in Figure 2.3.

The crux of synchronization is to align the sampling instances of the sensor nodes in a network. The offset created due to the drifting of the clocks needs to be corrected at certain intervals. If we take an example of a clock that guarantees a frequency stability of ± 20 Parts per million (ppm) , the maximum clock drift at the end of a one-second period is $40 \mu s$. A simpler synchronization problem would be to order the occurrence of events by comparing the local clocks. A more intricate problem would be to observe and record the drift and relative offset between the clocks. Reference broadcast synchronization (RBS) [7] and the flooding time synchronization protocol (FTSP) [8] are algorithms that adopt drift compensation techniques. Even though these protocols assure compensation of the software clock of a node, accurate synchronous sampling is not guaranteed. Offset, drift and jitter are a few common errors related to synchronization [9]. Along with these errors, the frequent trigger for synchronization also affects the reconstruction of the acquired signal. The clock offset introduces a constant time shift in the sampling instants. If the clock drift is considered to be constant over shorter periods of time, the

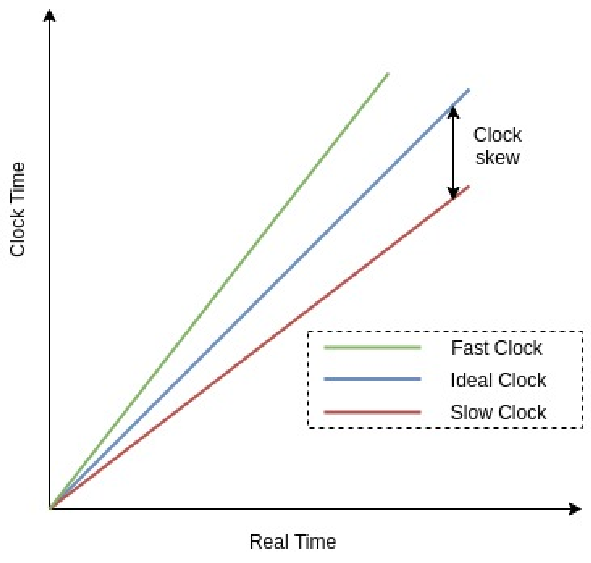


Figure 2.3: Clock Skew illustration [6]

acquired signal will phase away from the original signal proportional to the drift rate. The clock period jitter introduces non-linear distortion to the signal but usually at a very insignificant level compared to the other two factors.

The work of Funck and Gühmann, discusses two time-synchronization methods, namely, proactive and reactive sampling. Proactive synchronized sampling makes use of *a priori* time synchronization, whereas reactive sampling acquires samples independently from their local clocks asynchronously. These samples may not be sampled at the same instant across all nodes. This requires interpolation and re-sampling at individual channels for synchronous output. A hybrid approach also exists, in which the clocks synchronize to a master clock and a *posteriori* synchronization is performed on the obtained timestamps. The post-processing synchronization is also called timescale transformation, wherein, the local timestamp of the sample at one sensor is mapped onto the local time of another sensor node [9]. However, this approach adds communication and computational overhead since, the timestamp of each sensor node must be transmitted along with the data.

According to Feng [11], a delayed measurement start-time, sampling frequency instability and sampling interval jitter between sensor nodes are also key sources that lead to non-synchronous sampling in sensor networks, apart from clock synchronization error. Reconstruction of signals in the time-domain being a computationally complex task, is replaced by using a frequency-domain correction approach to recover the true spectral density of the asynchronous samples in this work [11]. The power spectral density is obtained using fast Fourier transform (FFT) method. The corrected spectral density and correlation functions are then used in modal analysis algorithms.

The software for the sensor nodes must be compact and efficient due to the lack of room for extensive computational power and memory on the microcontrollers. A modular design without much inter-dependency would enable to easily extend and improve the application. This requires the need for a flexible software architecture that is broken down into different modules for each function [12]. Efficient interrupt handling in an embedded system is important to avoid latencies and power overhead [13].

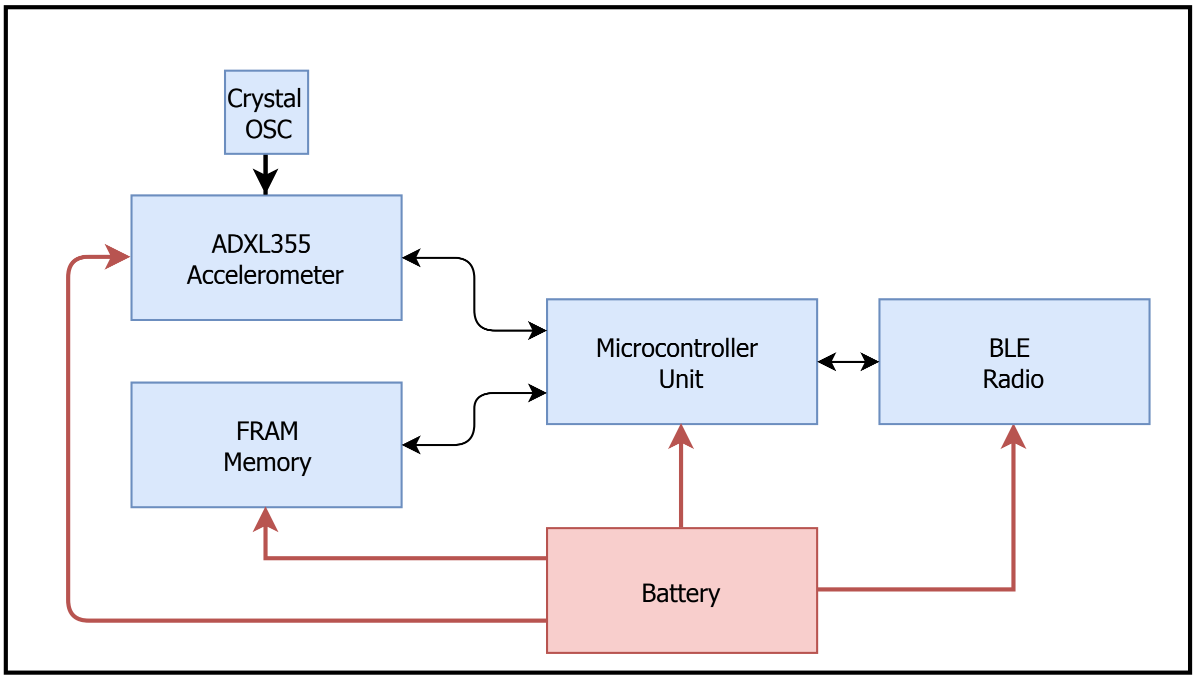


Figure 2.4: Hardware architecture of the sensor node

2.4. Hardware System: Sensor Node

The sensor node consists of various commercially off the shelf components based on the requirements of the application. The Printed circuit board (PCB) with the sensor, memory, microcontroller, radio and battery were already designed in the work of Zeo. The hardware architecture of the node is represented in Figure 2.4. The dimensions of the board are 27 mm x 24 mm. A brief description of each component in the sensor node is described in the following sub-sections. Section 2.4.4 on the crystal oscillator is the new addition to the already existing sensor node layout.

2.4.1. Accelerometer

The accelerometer ADXL355 from Analog Devices [15] was suggested for this research project by the stakeholders at ASML. This is a tri-axial MEMS based accelerometer with digital output. It also has an integrated temperature sensor. This device can be used as an inertial measurement unit (IMU) in stabilization systems, structural health monitoring, tilt sensing, robotics and condition monitoring applications. Together with a small form factor and low power consumption, this device is ideal for implementation in sensor networks. The ADXL355 includes three high resolution $\Sigma - \Delta$ Analog-to-digital converters (ADCs) to give digital output that is insensitive to the fluctuations in 1.8V supply voltage. Filters are included before and after these ADCs to provide accurate output. The sensing elements for the three axes have separate signal paths to reduce the offset noise and drift. The range, output data rates and filter corner frequencies are user-programmable via register access. The accelerometer has provision for a communication interface with the host via Serial peripheral interface (SPI) or Inter-integrated circuit (I^2C) protocols. More information on the accelerometer can be found in Section 2.6.2.

2.4.2. Memory

The memory used here is a Ferroelectric random access memory (FRAM), which is nonvolatile thereby eliminating complexities, overhead and bus delays as opposed to Electrically erasable programmable random access memory (EEPROM). The FRAM with part name CY15B104Q from Cypress [16] was selected based on its promising features. The non-volatile data storage is combined with the high

RAM performance. Since the data becomes non-volatile as soon as it is written, there is no data loss even during power disruption. The FRAM uses a mere $300\mu A$ active current and $100\mu A$ standby current owing to low power consumption. In addition, fast read/write cycles, high endurance and a guaranteed data retention period of 151 years is also offered. It supports two SPI modes with a speed of up to 40MHz.

2.4.3. Wireless Microcontroller with Radio

Bluetooth low energy (BLE) wireless communication standard was selected based on its low power consumption and output current while it also offers considerably decent data rate options for the application. The peak current generated must be low since the system is operated by a battery with low current handling capacity. The link quality and throughput can be improved by making use of the modulation schemes of BLE. The latest microcontrollers also offer built-in radio module. The CC2640R2F microcontroller with a BLE radio from Texas Instruments [17], was chosen since it offers both Cortex-M3 and Cortex-M0 enabling higher processing capability while keeping the transmission power and current consumption at a minimum of 5 dBm and 6.1 mA respectively.

2.4.4. Crystal Oscillator

The sensor requires an oscillator with a stable frequency of 1.024 MHz. A quartz crystal based oscillator is known to provide greater frequency stability compared to a simple LC or RC oscillator. The frequency stability is characterized by its ppm value and can be expressed as:

$$ppm = \frac{\Delta f}{F} \tag{2.1}$$

where F is the center frequency and Δf is the change in frequency over varying temperature and time. There are different types of crystal oscillators such as simple crystal oscillator (XO), temperature compensated oscillator (TCXO), Oven controlled oscillator (OCXO) and voltage controlled oscillator. The current consumption of crystal oscillators is usually in the range of milli amperes. A voltage supply of 3.3V is provided by the battery source. The form factor of the oscillator is also an important factor for our application. Simple XOs are usually smaller than the other oscillators while also promising a tolerable frequency stability. For this application, a simple crystal oscillator could fulfill its purpose. While choosing a crystal oscillator, there are a number of features to be looked into such as the frequency range, tolerance over temperature and ageing, operating voltage and form factor. A comparison of the available crystal oscillators that are suitable for our application are presented in Table 2.1.

Table 2.1: Crystal oscillator comparison

Part Number	**Frequency Range(MHz)**	**Operating Voltage(V)**	**Frequency Tolerance(ppm)**	**Dimensions**
Si510/511	0.1 to 250	3.3, 2.5, or 1.8	±30	2.5 × 3.2 mm
GXO-3201L	0.75 to 50	2.5 - 3.3	±25	2.5 × 2.0 mm
IDT XL	0.75 to 1350	2.5 or 3.3	±20	3.2 × 2.5 mm

2.4.5. Battery

The requirements of the application specify a battery run time of 48 hours. There is also a constraint on the maximum power dissipation of 10 mW. Hence these factors were taken into account when choosing the battery. The battery must be able to handle the current consumed by the components of the sensor system. One such battery that satisfies most of the criteria is the $Li-SOCl_2$ based LTC-5PN. This battery also offers good voltage stability with a small form factor. The battery is attached onto a separate PCB mounted on the board with all the other components. This board is replaced after the battery capacity has been depleted.

2.5. Network Synchronization

The three main objectives of the wireless communication between the sensor and sink nodes are to enable clock synchronization, an event-triggered interrupt and data retrieval. The data retrieval of the samples does not have a strict time constraint, whereas the clock synchronization between the nodes is pivotal for the wireless network. The communication protocol aims at reducing the delay in communicating the start of an event across nodes by providing a common notion of time between the network clocks. The different states of the system are initialization, idle, measurement and data collection.

The channel access for the network is provided using Time Division Multiple Access (TDMA) mechanism. The time frame is divided into a number of slots depending on the number of sensor nodes present at the system startup. Each sensor node is assigned a particular slot number to reply to the beacons sent by the sink. This enables the sink to keep track of the sensor nodes in the network. Since the network is to be placed inside the Wafer handler with a metallic closure, the effects of external network interference, multi path fading and electromagnetic interference were taken into account when designing the communication protocol.

The synchronization of sensor nodes must remain within certain limits in spite of the aforementioned factors. In the network, we consider the sink node to be the master clock which transmits the beacon periodically to all sensor nodes. Upon reception of the beacon, all the clocks of the sensor nodes synchronize with the master clock. This is done by estimating the local clock offset from that of the sink's clock. The interval in which the sink sends a beacon is calculated by considering the tolerance of the Microcontroller unit (MCU) clock crystal oscillator. The data sheet specifies the tolerance of the MCU crystal and real time clock crystal to be 20ppm. For a required accuracy of $55.55\mu s$, this interval is 1.389 seconds. In the initialization and idle states, the network is kept synchronized by periodic transmission of beacons and reception of beacon replies. This is also useful in keeping a check on lost connection between the nodes and sink.

The user can start the measurement after the initialization. There can be two events that trigger data collection, namely, the machine stopping suddenly and an overshoot of the acceleration threshold. These events are communicated to the network via the beacons from the sink nodes. The data collection stops once the machine stop event is identified and will continue to sample for the next 30 seconds after the acceleration threshold overshoot event is registered at all sensor nodes through the sink beacons. Data retrieval starts when the required data is placed in the memory with time stamps. The data acquisition process is described in detail in Section 3.3. The previous work concludes with an operational Bluetooth low energy network with a synchronization accuracy of $4\mu s$ between the sensor nodes. This is achieved by sending periodic beacons every second from the sink to sensor nodes with the timestamp of the sink. The MCU on sensor node calculates the drift and compensates for the offset. The power consumption of the components of the system were also examined in different modes of operation.

2.6. Accelerometer Sensor

An accelerometer can be represented as a mass-damper-spring system. The most common accelerometers are either based on piezoelectric effect or capacitance. Other sensing techniques include electron tunneling, resonance, optical and thermal based sensors [18]. Piezoelectric accelerometers depend on the stress generated by the force on the micro-crystal structures, thereby generating a voltage. Capacitance based accelerometers sense the variation in capacitance due to the movement of the plates. The latter is used in our project and the following subsections elaborate more on it. They are called MEMS accelerometers.

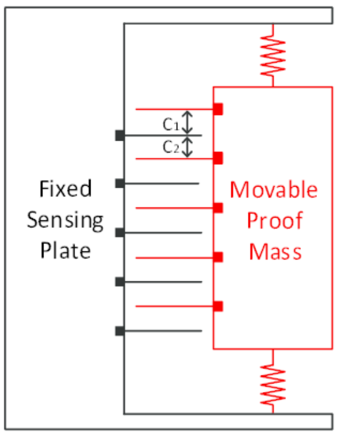


Figure 2.5: MEMS accelerometer operating principle [19]

2.6.1. Operating Principle

Micro-electromechanical systems (MEMS) are components manufactured using micro-fabrication technique which contain small mechanical parts such as membranes or cantilevers. MEMS are mostly used for miniaturizing devices to adapt to space-constrained applications. They are also more energy efficient than other types of accelerometers. MEMS are directly integrated with the electronics on a silicon chip which eases the process of system integration.

As mentioned above, MEMS accelerometers are based on change of capacitance due to the disturbance of micro-mechanical structures within the sensor. For a parallel plate capacitor,:

$$C_0 = \epsilon_0 \epsilon \frac{A}{d} = \epsilon_A \frac{1}{d}, \tag{2.2}$$

where ϵ is the permittivity constant of the material, A is the area of electrodes, and d is the distance between the plates. The capacitance varies with respect to each of these parameters which is the main principle behind MEMS sensing. A proof mass suspended through a mechanical structure moves between the movable and stationary plates of the capacitor as seen in Figure 2.5. The proof mass shifts down when the acceleration vector points upwards [19]. This shifts the plates thereby inducing a change in capacitance. The displacement of the proof mass and its output voltage are calculated with the measured capacitance difference due to their linear proportionality. The proof mass weighs approximately 0.1μg [20]. To obtain accelerations in three axes, sets of capacitors are placed in perpendicular directions. The high stiffness of the springs and the lightweight proof mass contributes to the high resonant frequency of MEMS sensors.

The output of the MEMS accelerometer can either be analog or digital. Analog accelerometers provide a continuous output proportional to acceleration. Its digital counterpart makes use of pulse width modulation, wherein, the width of the pulse at high voltage is proportional to the acceleration. The range of the accelerometers can be from $\pm 1.5g$ to $\pm 16g$. For our application of condition monitoring which involves sudden stops and starts, we require a range around $\pm 2g$. For a given input to the accelerometer, the magnitude of the response signifies its sensitivity. The higher the sensitivity, the better the accuracy of the acceleration output is. The sensitivity and noise of an accelerometer impact its resolution. The operation and performance of the ADXL355 sensor under consideration is discussed in the following sub-sections.

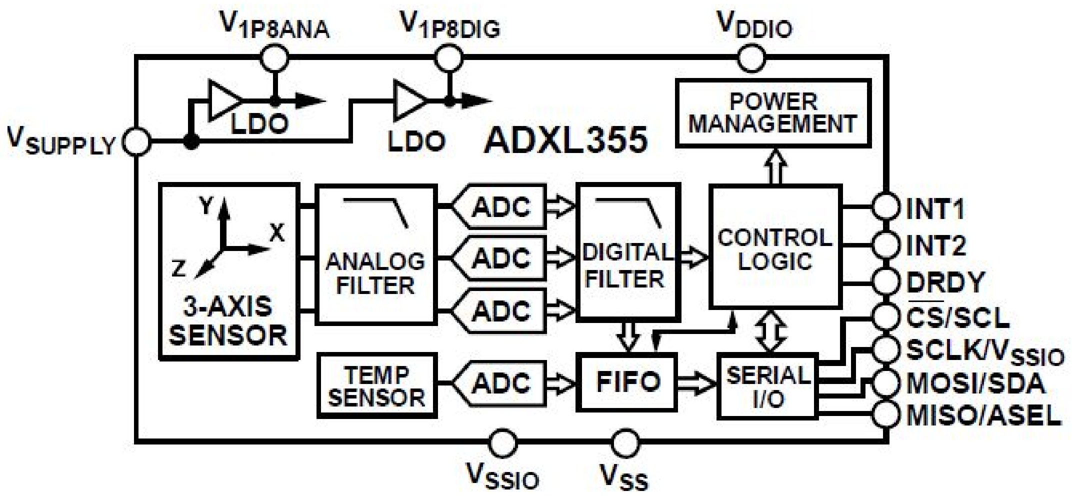


Figure 2.6: ADXL355 functional block diagram [15]

2.6.2. ADXL 355 MEMS Accelerometer

The ADXL355 presents acceleration measurement along three perpendicular axes, which was briefly introduced in Section 2.4.1. The important features of the sensor is shown in Table 2.2. The accelerometer provides three user-selectable ranges with the respective sensitivities as indicated in Table 2.3. It weighs 260 milli grams and measures of 6 mm x 6 mm x 2.1 mm in dimension. The ADXL355 also has a temperature sensor, but this feature will not be used for the application under consideration. A functional block diagram of the sensor is illustrated in Figure 2.6.

Table 2.2: ADXL355 Parameters

Parameter	**Description**	**Min**	**Typical**	**Max**	**Unit**
F_s	Sampling frequency	3.9		4000	Hz
$N.D$	Noise density		25		$\mu g\sqrt{Hz}$
V_{supply}	Power supply	2.25	2.5	3.6	V
T	Temperature range	-40		+125	$^{\circ}C$
I_{meas}	Current consumption during measurement		200		μA
$I_{standby}$	Current consumption at stand by		21		μA

There are three sets of MEMS sensing elements that follow separate differential signal trajectories to reduce noise and drift. The ADXL355 being a digital variant of a MEMS accelerometer, it includes three ADCs to provide high resolution 20-bit digital output per axis. The analog anti-aliasing filter, filters the X, Y and Z axis analog outputs before entering the high resolution ADCs. A successive approximation register(SAR) ADC is used by the temperature sensor to provide 12-bit resolution digital output. The $\Sigma - \Delta$ ADC used for acceleration sensing has an oversampling architecture. Oversampling reduces aliasing and quantization noise. Noise shaping is an additional feature of the $\Sigma - \Delta$ modulator

Table 2.3: Range and sensitivity

Range(g)	**Typical Sensitivity**(LSB/g)	**Scale factor** $(\mu g/LSB)$
± 2.048	256,000	3.9
± 4.096	128,000	7.8
± 8.192	64,000	15.6

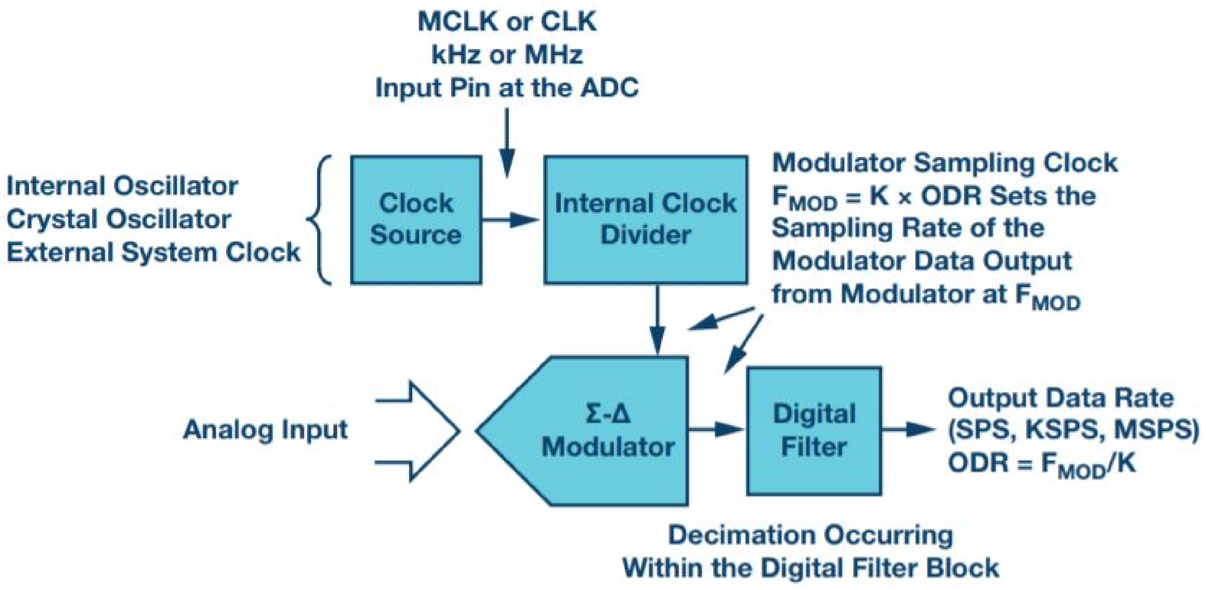


Figure 2.7: Signal flow via $\Sigma - \Delta$ ADC and Digital filter [21]

that shifts the quantization noise of the ADC into high frequency while a low pass filter removes this noise [21].

A flow diagram of a typical $\Sigma - \Delta$ ADC is given in Figure 2.7. The ADC operation runs on an external or internal clock source and is divided into the modulator frequency. This is the rate at which the digital filter receives data from the modulator. The digital filter is a band-pass filter, the high and low-pass poles of which can be accessed and modified via filter settings register. Through the second stage of low-pass decimation filter, the desired output data rate between 3.9 Hz to 4 KHz is obtained. This decimation happens inside the digital filter. There is a delay introduced with the decimation filter setting and attenuation occurs at a corner frequency of Output data rate (ODR)/4. For an ODR setting of 1KHz and corner frequency at 250 Hz, the group delay is 1.78 ms. If an interpolation filter is also enabled, then for these settings the total delay adds up to 2.75 ms. The total group delay is measured from the analog input to the ADC until the acceleration data is ready to be read at the interface. The resonant frequency of this model is 2.4 KHz.

The sensor has provision for data transfer via SPI (Serial Peripheral Interface) or I^2C (Inter-integrated circuits) protocol. I^2C is implemented for low-speed peripherals for on-chip signal transfer. It has a master-slave architecture with multiple masters and slaves. Each slave device is identified by a unique address. The devices are connected to the masters using only two lines which are data line and a clock line. SPIs are widely used in embedded systems for short distance communication. This allows any number of slave devices for a single master to be selected through the slave select (SS) line which is a active low signal. Whenever there is data transferred between a slave and a master,this line is set low to indicate that the bus is occupied by a particular slave device. There are separate data lines and clock lines to enable full-duplex transfer while I^2C offers only a half-duplex. SPIs offer speed over a broader range which is much faster than I^2C. Hence SPI is preferred for most sensor network applications where the data transfer speed cannot be traded-off with the amount of data. The SPI clock speed can range from 100 KHz to 10 MHz with the clock polarity and clock phase being 0.

The sensor data is formatted in two's complement with the most significant bit first. The newest data from the three axes are stored in the respective data registers. Whenever there is a new set of acceleration data available, the DATA_RDY bit of the STATUS register is set. The data acquisition from the registers is described along with software description in Section 3.3.

Performance

Even when the accelerometer is still, there is an offset in the average value of the signal. This is the sensor bias which is usually specified in the data sheet. The smallest acceleration that can be detected by the sensor is its resolution. However, the resolution is limited by noise. For a digital accelerometer, the resolution(R) is given by:

$$R = \frac{Range}{2^{ADCbits}} \tag{2.3}$$

For our application we use the ±2.048 g range with the 20 bit ADC. From the equation 2.3, the resolution for these parameters gives 3.906 μg.

Noise is any deviation from the actual signal. The power spectral density specifies the noise level of the sensor above which the signals can be measured. Usually, additive white Gaussian noise (AWGN) is assumed in which there is equal distribution of noise in the frequency domain and a normal distribution in the time domain. The Root mean square (RMS) and peak-to-peak noise are deduced from the power spectral density (PSD) plot with a 6σ approximation. The main sources of noise in MEMS sensors are thermal noise due to mechanical vibrations and $1/f$ electronic noise and quantization noise from signal conditioning. The mechanical noise arises due to Brownian motion of gas molecules. A trade-off exists between bandwidth and noise since noise is distributed across all frequencies. The RMS noise is given by:

$$n_{RMS} = \rho\sqrt{1.6B} \tag{2.4}$$

where ρ is the noise density of the sensor given as 25$\mu g/\sqrt{Hz}$, and B is the cut off frequency set in the filter settings register.

For an ODR setting of 1 KHz, the LPF (low pass filter) frequency is 250 Hz. The factor 1.6 represents the quality of the filter. This is the estimated value for a single-pole low pass filter. Applying these values to Equation 2.4, we calculate the RMS noise to be 500 μg. The ADC quantization noise exhibits a Gaussian nature. The peak-to-peak noise is calculated as 6.6 ×RMS noise spanning the 99.9% of the signal [22]. In this case it is 3.3mg. Since the peak-to-peak noise is significantly larger than the resolution, there is a high limiting factor on the acceleration measurement. The measurements are guaranteed to be stable over a large temperature range, making the sensor ideal for applications that demand temporal and temperature stability.

2.6.3. Sensor Synchronization

The accelerometer, in normal operation, employs the internal clock to run the functional components of the sensor. The frequency of internal clock of the sensor is 4 KHz which is multiplied to provide the required 1.024 MHz. The frequency variation of this clock is as given in Figure 2.8. The clocks have a frequency variation of 20% from the center frequency.

The signal representing the output data rate (ODR) is the alignment of the output of the decimation filter. When the sensor operates with its internal clock, the data is retrieved asynchronously by the processor. The 20% drift of the clock also shifts the sampling instant temporally leading to output signal distortion. However, if there is a need for absolute synchronization with the external sources, this internal clock can be over-ridden by external synchronization. The ADXL355 sensor offers two possibilities for synchronizing the sensors externally.

External Synchronization with Interpolation

In external synchronization with interpolation method, an external signal corresponding to the required output data rate is fed to the sensor. The time resolution of the interpolation filter is 64 times the

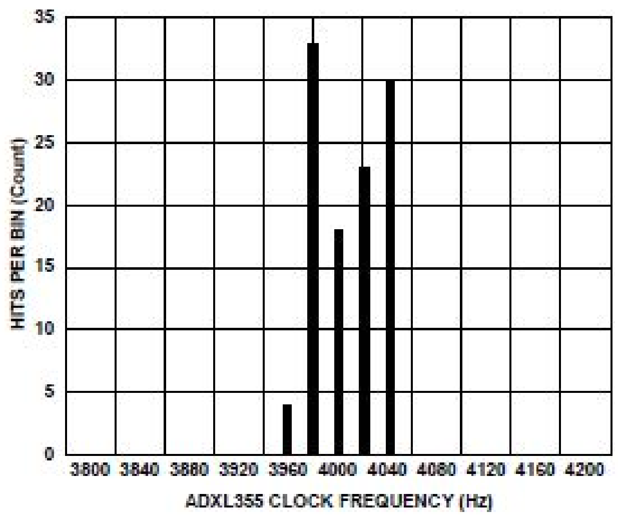


Figure 2.8: Internal clock frequency distribution of ADXL355 [15].

sampling frequency. This forces the sample to be ready at the desired rate which is asynchronous to the sensor's clock. Nevertheless, this introduces an increase in group delay and attenuation. The delay between the sync signal input and the data ready output for 1 KHz ODR is 14 internal oscillator cycles which is close to 3.5 ms. A minimum of four samples per second could be lost in this technique.

Full External Synchronization

The other option is to provide full synchronization with an external master clock and a synchronization signal. This method does not include an interpolation filter thus already eliminating the oscillator cycle delay. The external clock that is provided over-rides the internal oscillator to become the master clock for the sensor. All the functional blocks of the sensor now runs with this external clock as the master. Additionally, a synchronization signal is required to align the output of the decimation filter with a particular clock edge. This technique is useful in sensor networks that require data to be sampled synchronously at all sensor locations. The external signals required to the sensor are illustrated in Figure 2.9. When the sensor is to be used in this mode, the following conditions must be met:

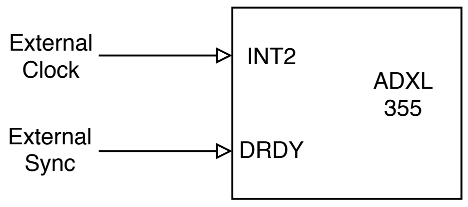


Figure 2.9: External clock and synchronization signal input to sensor

- The external master clock must be 1.024 MHz with a low frequency tolerance
- An external synchronization signal must be provided to align the internal output data rate filters

- The width of the external synchronization pulse must last for at least four times the external clock period.

If the external synchronization signal is missing, the sensor runs on its own synchronization. The external clock is given to the INT2 pin (pin 13) and the synchronization signal is supplied to the DRDY pin (pin 14) of the sensor. The samples that are collected before the synchronization signal arrives, is discarded. This signal realigns the ODR and the corresponding filter in order to achieve synchronous sampling. As a consequence, two subsequent samples are corrupted which is investigated in Section 4.2.3. The implementation of this mode is described further in Section 3.5.

3

Synchronization Approach and Realization

3.1. Overview

This chapter discusses the approach followed for the synchronization at the sensor level. The firmware architecture for the sensor node is described in the following section 3.2. Data acquisition from the accelerometer and its processing is briefly explained in Section 3.3. The two techniques that can be used to obtain synchronous sampling are presented in Section 3.4. The implementation of the two approaches are detailed in Section 3.5.

3.2. Embedded Firmware Architecture

To properly understand the hardware, it is important to determine the hardware dependency of an embedded system. The different hardware peripherals involved in this system were discussed in Section 2.4. In order to access and control the various hardware resources such as timers, memory, sensors, serial communication etc., device drivers which comprise the so called hardware abstraction layer (HAL) are required to be developed [23]. The device drivers act as an interface between the hardware and the embedded software. They are usually modelled around the peripherals. In this system, a separate device driver is implemented for each of the peripherals such as memory, sensor and SPI to facilitate data transfer.

The sensor node software makes use of the Texas Instruments driver library functions for register access of the micro-controller rather than using the real-time operating system (RTOS) in order to achieve micro-second level network synchronization. The software architecture of the system is given in Figure 3.1. While the grey-colored blocks were already implemented in the previous project [14], the communication protocol and coordinator blocks have been left for future work. The blue-colored blocks were modified or partially developed. The white-colored blocks were entirely implemented for this thesis. The red-colored blocks represent the peripherals of the BLE microcontroller unit. A brief description of the software drivers that were implemented are described in the following sub-sections.

3.2.1. SPI Driver

The serial peripheral interface (SPI) handles the communication between the host processor and other peripherals. SPI is used in our system since we have the requirement for high data rate during data transfer between peripherals. The SPI bus lines are routed using general purpose input/output (GPIO)

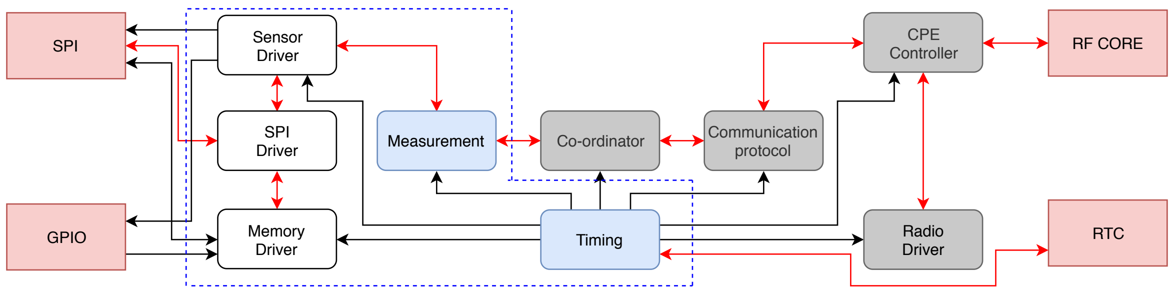


Figure 3.1: Sensor node Embedded Firmware Architecture (*Highlighted blocks are the focus of this thesis*)

pins for SPI operation. In Figure 3.2, the SPI interface between the sensor and memory functioning as slave devices for the single MCU master is illustrated. The SPI operates with only four wires, thus making it an easier choice in most embedded system development. The serial clock (SCLK) drives the data synchronization between the master and slave devices. The master device provides the SCLK. The throughput of the serial data depends on the serial clock speed. The clock speed used here is 2 MHz. The data lines are the master-out slave-in (MOSI) and master-in slave-out (MISO) which enables full duplex communication. The slave-select or the chip-select ($\overline{CS}$) line is an active low signal.

Since our system has two slave devices, there are two $\overline{CS}$ pins connected via he GPIO to the master MCU. The corresponding $\overline{CS}$ line is pulled down or asserted according to the slave that has access to the SPI bus for data transfer. Only one slave can have access to the SPI bus at a time. The master has control over the slave's access to the SPI bus. The data is sent as bytes with the most significant bit first, over the SPI bus lines for each transaction. The $\overline{CS}$ line is de-asserted after the slave has finished transferring its data. At start-up, the SPI bus is initialized by mapping the GPIOs to the SPI signals. The SPI in the MCU is configured in the master mode and specifies the clock polarity and phase as low (mode 0). A few dedicated application program interfaces (APIs) were written to carry out basic SPI functions such as *Spi_Send, Spi_Receive, Spi_FlushFifo* and *Spi_Transaction*.

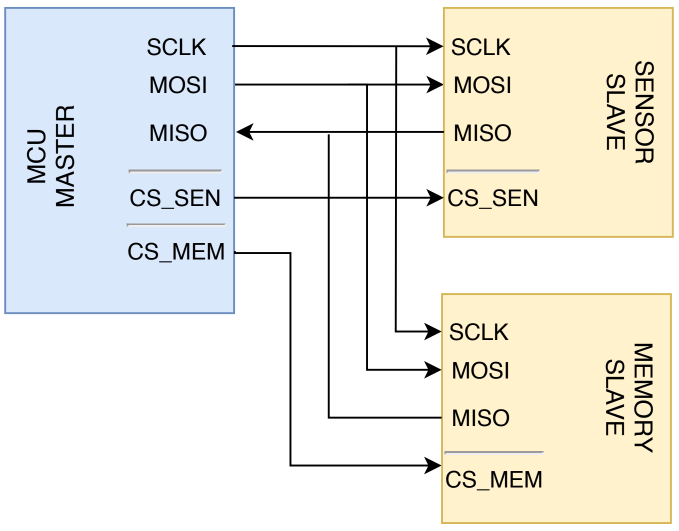


Figure 3.2: SPI configuration for memory and sensor

3.2.2. Sensor Driver

The ADXL355 sensor uses the SPI bus to communicate with the system. The sensor, being one of the SPI slaves, requires an interface between the MCU and the sensor for data transfer, register access and control. The sensor driver implements these functionalities in its respective Application programming interfaces (APIs). The internal registers of the sensor offers controlled access in order to program, modify or read certain parameters pertaining to its operation. The reading and writing of these registers are facilitated through the APIs *Sen_Read_Register* and *Sen_Write_Register* respectively by passing the required parameters of register address, destination address to store the read value and value to be written. The data sheet [15] provides a complete list of control registers with its corresponding addresses.

The GPIO pin for the SPI chip select is specified in a macro. The *Sen_Init* API configures the registers required before we start the measurement. This includes resetting the sensor, setting the output data rate (ODR), specifying the acceleration output range in gravitational(g) unit and selecting the mode of synchronization. The sensor is configured to sample at a frequency of 1 KHz with a measurement range of $\pm 2g$. The power control register is then set to measurement mode to start the normal operation of the sensor. The necessary sensor interrupts are also mapped to the GPIO of the MCU before starting to read the data. The most important API of this driver is the *Sen_Read_Accelerometer*, which is responsible for reading the STATUS register and the three dedicated data registers for each of the x, y and z axes. The STATUS register is a read-only register whose DATA_RDY bit is set, when a complete x, y and z-axis measurement is made and the results can be read out. The DATA_RDY bit clears on a read of the STATUS register. The acceleration data is left justified and formatted in two's complement. Each axis measurement is 20 bits long and stored in three registers of 8 bits each. The combined data is returned as a data array buffer from this function.

3.2.3. Memory Driver

The memory peripheral used in the system also makes use of the SPI for data transfer. The FRAM from Cypress being a 4 Mbit dense memory, requires a three-byte addressing during an SPI transaction. The FRAM accesses the SPI bus when the $\overline{CS\_MEM}$ is asserted. There are six opcodes that provide control over the FRAM device such as write enable/disable, write/read data and write/read status register. Each read/write cycle has an opcode, 3-byte address and data specified in that order on an SPI bus with an active low $\overline{CS}$ line as an 8-bit data transfer. The APIs that aid the operations are implemented as part of the memory driver. They are *Mem_Write, Mem_Read* and *Mem_ReadStatus*. The first address location of the memory array is allocated to the memory flag. The flag is set when the memory is filled with 30 seconds worth of data, and once the memory is read out, this flag is cleared.

3.2.4. Timing

The software timers that generate events for the timeouts and periods were written with the real time clock (RTC) as its base. The timeouts and periods were kept track of by counting system ticks. Each system tick is approximately 1 milli second. The modules were used to perform certain start-up functions such as initializing the real time clock, synchronizing the RTC with the radio timer (RAT) during wake up and enabling channel 2 of the RTC to generate periodic signal. The dynamic functions of the timing module are meant to initialize and update the periods and timeouts during run-time. For example, a timeout is enabled to detect the synchronization pulse within a time interval.

3.2.5. Other Modules

The command packet engine (CPE) controller enables the communication between the main processor and the RF(radio frequency) core by submitting instructions via register access. The radio driver makes use of the CPE controller to send instructions to the RF core to perform operations such as turning on/off radio, transmitting/receiving packets, setting data rate, transmission power and frequency channel.

The 4 MHz radio timer in the RF core schedules these instructions. A partial development of the communication protocol module was used to test the reliability of network synchronization and link quality between the sensor and sink nodes. The yet to be implemented coordinator module can act as a manager by sending commands to the memory and sensor drivers via the measurement block, handle power management, sleep wake-up cycles of the microcontroller, among others.

3.2.6. Interrupt Handling and Interrupt Latency

An interrupt is a signal from an external hardware or internal software event to the processor to stop the current flow of instructions and service the triggered events. In accessing the above described drivers, interrupts have been used to avoid missing out on a time-critical occurrence of an event. The interrupt handling properties affect the data acquisition process greatly. Interrupt latency is defined as the number of clock cycles that a processor takes to respond to an interrupt event from the time of its arrival. The Cortex-M3 processor used in our system takes 12 cycles from the assertion of the interrupt request up to the execution of first instruction in the handler. The processor offers hardware-supported entry and exit of the interrupts. The two main interrupts used in the architecture are for the synchronization pulse to the sensor and the data-ready signals generated from the ADC of sensor. The interrupt mapping between the sensor and the microcontroller is as shown in Figure 3.3.

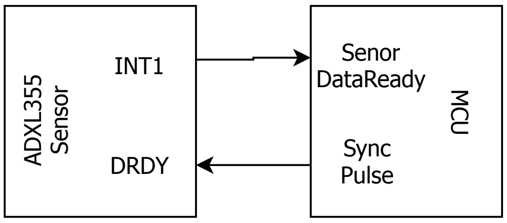


Figure 3.3: Interrupt mapping between the ADXL355 and microcontroller

The *INT1* (pin 12) and *DRDY* (pin14) of the ADXL355 sensor are mapped to the two GPIO pins of the microcontroller. The *Sensor Data-Ready* GPIO pin is mapped to receive the interrupt from *INT1* pin. Whenever the synchronization pulse is to be fed to the sensor, the *DRDY* pin receives it from the Sync Pulse GPIO pin. Since the interrupt service routines perform critical tasks, they must be kept short to reduce any latency that they may introduce to the flow of software control.

3.3. Data Acquisition

As the name suggests, data acquisition is the process of acquiring and storing measurement data for further analysis. Retrieval of data can be continuous, event-driven or on-demand. The sink generates the on-demand data transmission. For continuous retrieval, the data is sent periodically. In an event-driven query, the data transmission is triggered only in the case of an event. Obtaining a discrete signal in equal intervals from a continuous process is called sampling. This interval is termed as the sampling rate of the signal. The sampling rate is determined using the Nyquist theorem, which states that

> *A continuous signal $x(t)$ can be uniquely identified by discrete samples $x[n]$ taken at an interval of $1/f_s$ seconds, if the sampling frequency f_s is at least twice the highest frequency component of the signal.*

The Nyquist frequency is half of the sampling frequency f_s. The above theorem holds true if the Fourier transform of the signal at frequencies greater than the Nyquist frequency is zero.

The data originating from the ADXL355 sensor cannot be directly used for modal analysis. The data must be obtained in units of gravity (*g*). The sensor outputs the acceleration as a 20bit ADC stored in three data registers per axis. The signal flow from the analog sensor to ADC output is described in Section 2.6.2. As an example, Figure 3.4 shows the steps followed to convert a 20 bit ADC into an acceleration value for the x-axis. The corresponding code snippet used is given in Appendix C.

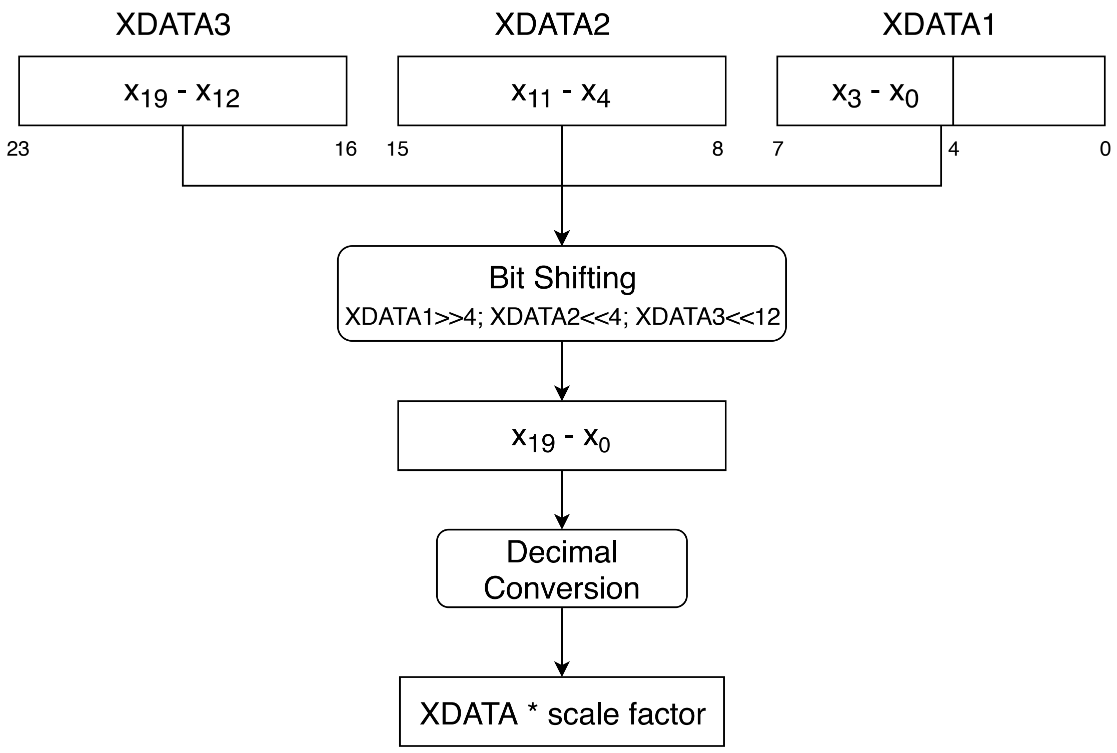


Figure 3.4: Data conversion flow for the x-axis measurement

Since the most significant bit of the data is first available at the ADC, the registers are read in the order of XDATA3, XDATA2 and XDATA1. The bits in the registers have to be shifted and combined to form a 20-bit value for each axis. The first 4 bits of the XDATA1 register are reserved and do not contain required data bits. Hence they are discarded when combining. The combined value is converted to decimal using the equation 3.1.

$$XDATA = -x_{n-1} * 2^{n-1} + \sum_{i=1}^{n-2} a_i 2^i \tag{3.1}$$

where n=20 bits of the number x. The combined XDATA is then multiplied by the scale factor, which corresponds to the range setting for the accelerometer as presented in Table 2.3. The resulting acceleration value is in units of g. The data is now ready for further analysis in the frequency or time domain. A simple orientation test was done to ensure the correctness of the data as depicted in Appendix A. Figure 3.5 shows the acceleration data plot for a 30 second period. The data was obtained at a sampling rate of 1 KHz with a $\pm 2g$ range. The sensor was kept on a fairly flat surface in a temperature controlled clean-room with the z-axis pointing upwards (orientation in the green box in figure A.1). Hence the acceleration value is approximately $0g$ along the x and y axes, and $1g$ along the z-axis. The slight offset from their ideal values are due to the sensor bias which is briefly discussed in section 4.2.3.

The data is repeatedly read from the internal registers of the x, y and z axes. The memory module stores the samples until the data are retrieved externally. From the specifications of the application,

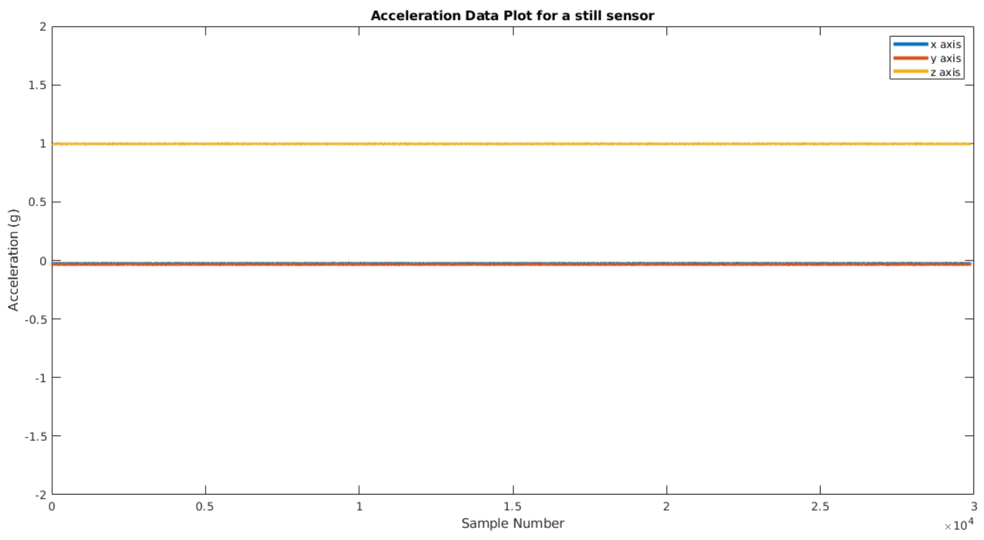


Figure 3.5: Acceleration data plot for 30 seconds

30 seconds of measured sensor data is made available for further processing. The non-volatile FRAM retains the data as long as required. The acceleration at all locations is sampled synchronously by the system. Depending on our application and network design, an event-driven method is deployed. The event can be the machine suddenly stopping or an acceleration overshoot, as discussed in Section 2.2. The measurement starts as soon as the user indicates it. But data retrieval is triggered only when encountering the above events. However, the sensor continues to measure at all times.

3.4. Synchronous Sampling

Time synchronization is crucial for wireless sensor networks that often limits the application's boundaries due to a demanding requirement for synchronous sampling. Combining and processing the measurements obtained across all nodes of a network is one of the greatest challenges. The knowledge of the relationship between the sampling instances of the measurement is crucial in achieving sensor-level synchronization. For the application of modal analysis, the sampling instance of the sensors must be synchronized.

Measuring the sampling time of a sensor can introduce two types of errors: Deterministic and random errors. The deterministic errors include the group delay of the sensor filters, propagation delay in the circuitry and clock offset. The random errors may consist of clock drift, interrupts, etc. These errors influence the synchronization accuracy of the measurement. In order to obtain synchronous samples, two different approaches are proposed: real-time and subsequent synchronization of samples.

Real-time Synchronization

In real-time synchronization, the sampling is triggered at the same time on all sensor nodes using synchronized clocks thus guaranteeing identical sampling instant. This method is illustrated in Figure 3.6. The generation of synchronization trigger signal is the most time-critical element of this method. Hence any compensation applied to the on-board MCU clock should not lead to timing jitter above the acceptable tolerance level. The propagation delay between the synchronization pulse and the subsequent triggering of an event must be kept minimal by employing interrupts.

From Figure 3.6, it can be observed that the synchronization pulse is given at the same time to all nodes. However, the detection of this signal at the sensor node might not occur at the same time as

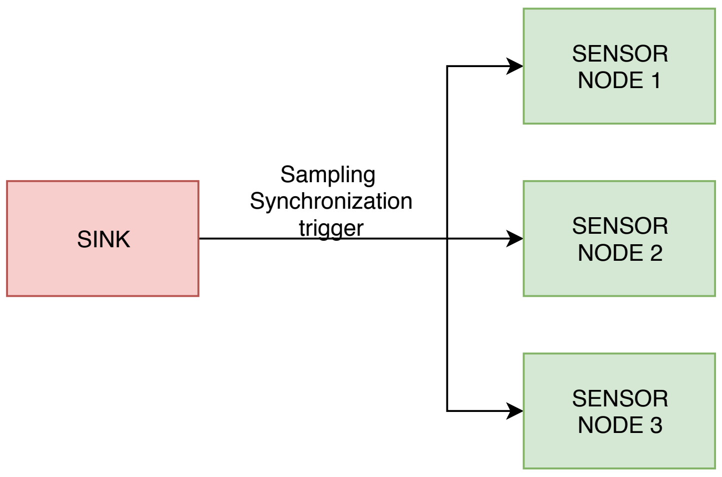


Figure 3.6: Real-time Synchronization approach in a network with 3 sensor nodes and a sink

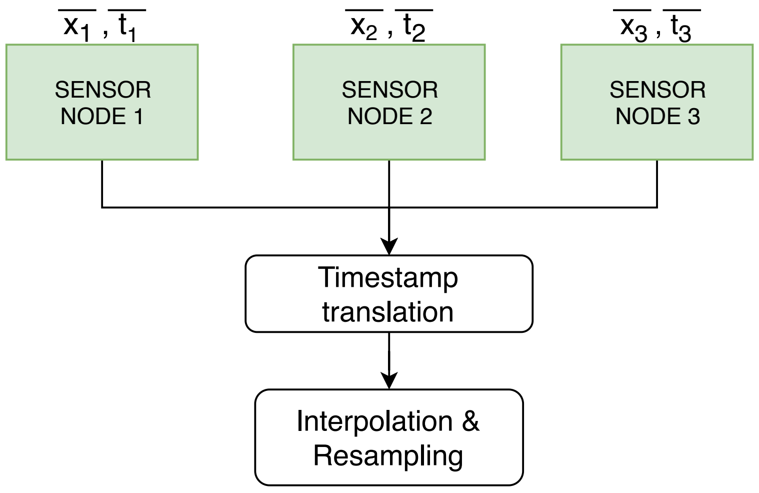

$\overline{x_1}, \overline{t_1}$
$\overline{x_2}, \overline{t_2}$
$\overline{x_3}, \overline{t_3}$
SENSOR NODE 1
SENSOR NODE 2
SENSOR NODE 3
Timestamp translation
Interpolation & Resampling

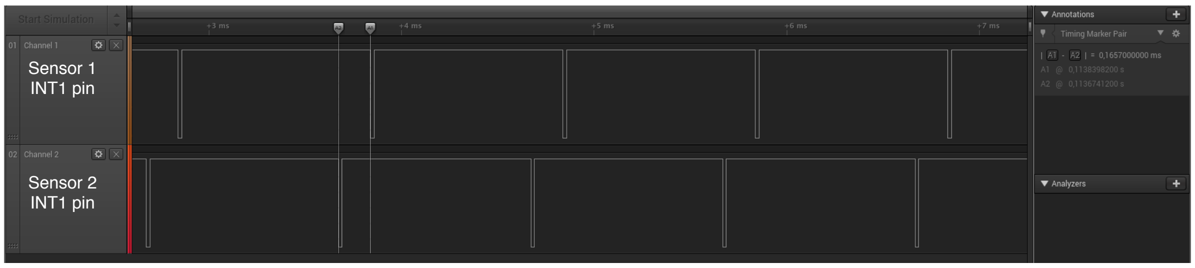
Figure 3.7: Subsequent synchronization approach in a network with three sensor nodes and a sink

the other nodes. This propagation delay places a limit on the maximum achievable synchronization accuracy. The advantage of this method is that there is no signal processing overhead at the sink node for synchronization. The obtained samples can be directly used for further analysis.

Subsequent synchronization

In subsequent synchronization, sampling is triggered with the sensor node's own local unsynchronized clock. The samples from different nodes would not have the same sampling times. The steps involved in this method are illustrated in Figure 3.7. Each sample is assigned a timestamp according to its corresponding local MCU clock before being sent to the sink. The challenge is to assign a sample timestamp that is considerably close to the real measurement time corresponding to the node's local clock. As a post-process, the local clock based timestamps are either translated to a global timescale of the sink node or to one of the sensor node's local clock. Further interpolation and re-sampling is performed to obtain synchronous samples when the sampling frequency satisfies the Nyquist theorem. The final synchronization accuracy is impacted by the time-stamping of the data.

This method also allows the sensor nodes to remain in low power state for a longer period of time. Considerable computational effort is required when post processing the timestamps for interpolation in order to obtain synchronous samples. The interpolation and re-sampling algorithms determine the



```
| A1 - A2 | = 0,1657000000 ms
A1 @ 0,1138398200 s
A2 @ 0,1136741200 s
```

Figure 3.8: Capture of the *INT1* pin of two sensors triggered when a data sample is ready to be read

synchronization accuracy of the sensor nodes.

3.5. Approach

The synchronization methods previously discussed, are to be realized for our application with the available resources. The developed embedded firmware (Section 3.2) provides the platform to implement the techniques. In order to quantify the synchronization accuracy between the sensor nodes, the interrupt signal that is triggered when a new sample is ready is used. Figure 3.8 shows a capture from a logic analyzer. The signals comprise an active-low ADC interrupt routed from the physical *INT1*pin of two different sensors nodes. The frequency of these signals corresponds to the output data rate of the sensor, which is 1 KHz. The sensor clocks are free running and unsynchronized. The trigger to start sampling is received at all nodes by the beacon transmission from the sink with the accuracy of the network synchronization protocol. From the previous work [14], this network synchronization tolerance is kept under 4 μs.

3.5.1. Real-Time Synchronization

The objective of synchronous sampling in this context is to ensure that the time difference between the falling edge of two corresponding *INT1* pulse from figure 3.8 is kept below a required minimum. In order to implement external synchronization on the accelerometer sensor, few steps described in Section 2.6.2 must be followed. The sensor must be fed with the necessary signals as given in figure 2.9. The generation of the two most important signals: the external clock and external synchronization signal are described in the following subsections.

Clock Generation

The external clock to be generated acts as the master clock for the ADXL355 accelerometer sensor. This clock drives the various functionalities of the sensor. The nominal frequency requirement is at 1.024 MHz as indicated in the data-sheet [15]. The final solution is to use a simple crystal oscillator (XO) as described in Section 2.4.4. However, in order to qualify the proposal, other clock sources were used to implement the synchronization. The clock sources considered were the general purpose timer offered by the microcontroller unit and a waveform function generator.

The CC2640R2F microcontroller has programmable timers that can be used as a timing resource. Out of the four general purpose timers, timer 2 is configured in Pulse width modulation (PWM) mode as a 16-bit down-counter for our application. The timer is derived from a 48 MHz clock. A code snippet of the PWM mode configuration that outputs a 1.024 MHz clock is listed in Appendix B. This source was used since it is directly available within the microcontroller to check for the possibility of synchronization and it is better than what the ADXL355 internal clock offers. The next source that was used is a waveform generator from Keysight 33500B series [24]. The waveform generator uses *trueform* technology and offers a low jitter of $40ps$, a low harmonic distortion of 0.04% and a high frequency resolution of $1\mu Hz$. The amplitude is set at 3V peak-to-peak. This source was used to emulate a crystal oscillator.

These sources must be validated in order to obtain reliable results from the sensor node. The clock source has a great influence on the operation of the sensor. The nominal frequency is the designed operational frequency of the oscillator. The variation of the output frequency from the oscillators's nominal frequency is termed as frequency stability. The frequency stability of a quartz crystal in an oscillator is an important property that depends on temperature, aging, supply voltage, vibration noise, etc,. This variation is expressed in parts per million (ppm) as:

$$\epsilon[ppm] = \frac{Outputfrequency - Nominalfrequency}{Nominalfrequency} \times 10^6 \tag{3.2}$$

Even when the manufacturer of the proposed crystal oscillators in Table 2.1 designs them for a frequency of 1.024 MHz, it can never stably output that frequency. A frequency meter can be used to gauge how much the clock varies from its nominal frequency. For our experiment, Keysight 53230A universal frequency counters [25] was used. The clock output from either of the sources is fed to the frequency counter and the values are recorded to evaluate the frequency accuracy. The frequency stability of the two sources measured using the method above are displayed in Table 3.1. The histograms of the frequency distribution around the offset and random error are given in Appendix D.

Table 3.1: Frequency stability

Source	**Max Frequency (MHz)**	**Min Frequency (MHz)**	**Mean Frequency (MHz)**	**Frequency Accuracy (ppm)**
General Purpose Timer	1.025707331	1.026094390	1.026063352	+1668.11
Function generator 1	1.024000292	1.024000291	1.024000292	+0.28445
Function generator 2	1.024000271	1.024000268	1.024000269	+0.26364

An eye diagram aids in understanding a signal's characteristics by retrieving parameters such as amplitude, time distortion, etc. An eye diagram is generated by overlapping multiple sweeps of segments of a signal in a high-speed sampling oscilloscope. The measurement equipment used here is the Keysight InfiniiVision MSO-X 3034T [26]. The triggering edge is set to both the rising and falling edge to capture the eye pattern. The misalignment of the rise and fall times is termed as timing jitter. The eye height is the difference between the low and high level of the signal. The eye width is the measure between the left and the right crossing point. The eye patterns of the general purpose timer clock and function generator clock are given in Figures 3.9 and 3.10 obtained from the oscilloscope.

By comparing the eye patterns, we can infer that the eye crossings are more stable in a function generator clock than that of the general purpose timer. The eye height is constant for a general purpose clock signal whereas, for a function generator signal, a ringing effect can be noticed. This ringing is due to the mismatch in transmission line impedance of the probe. It was observed that the ringing effect did not induce any unusual behaviour at the receiving module.

Synchronization Pulse generation

The other key signal is the synchronization pulse to be given to the *DRDY* pin of the ADXL355 sensor. This signal aligns the output data rate filter to prepare the sample and trigger the interrupt on the *INT1* pin as in Figure 3.8. Section 2.6.2 explains more about the internal working of the sensor. This signal is taken as an interrupt to the MCU when it registers a beacon reception from the sink node. In order to emulate this behavior, the synchronization pulse signal is also generated from a function generator. Since beacons at the network synchronization level are transmitted at a frequency of 1 Hz, the function generator is also set to provide a square waveform at 1Hz. Right after the beacon

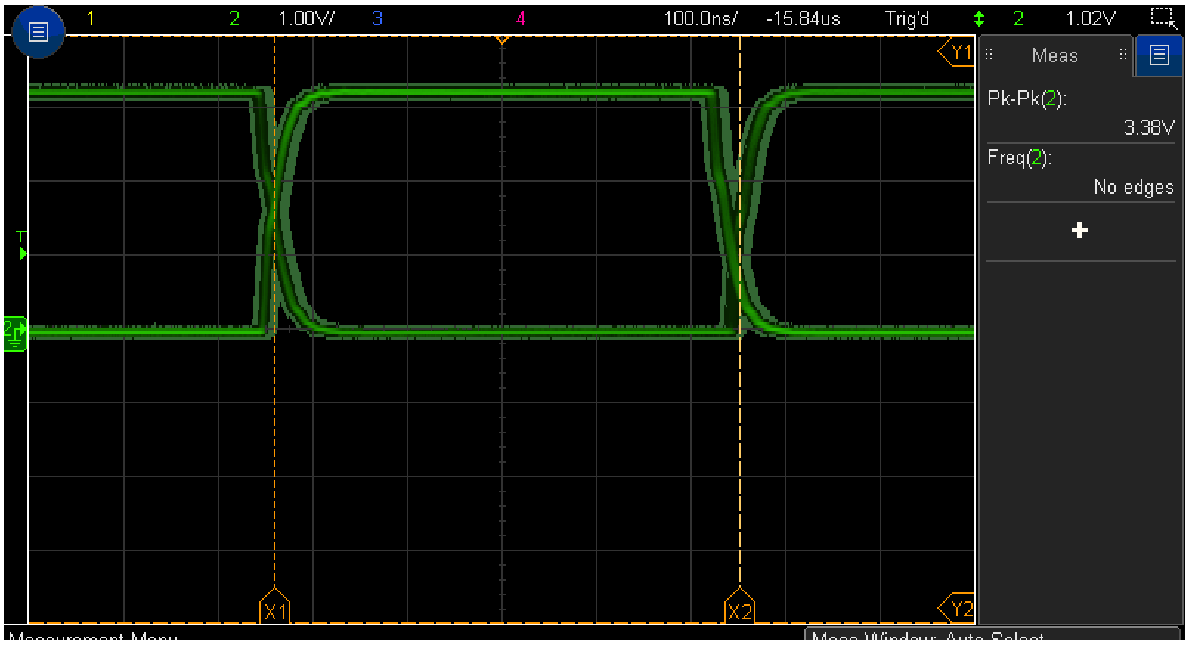


Figure 3.9: Eye pattern of the clock generated from general purpose time of the MCU

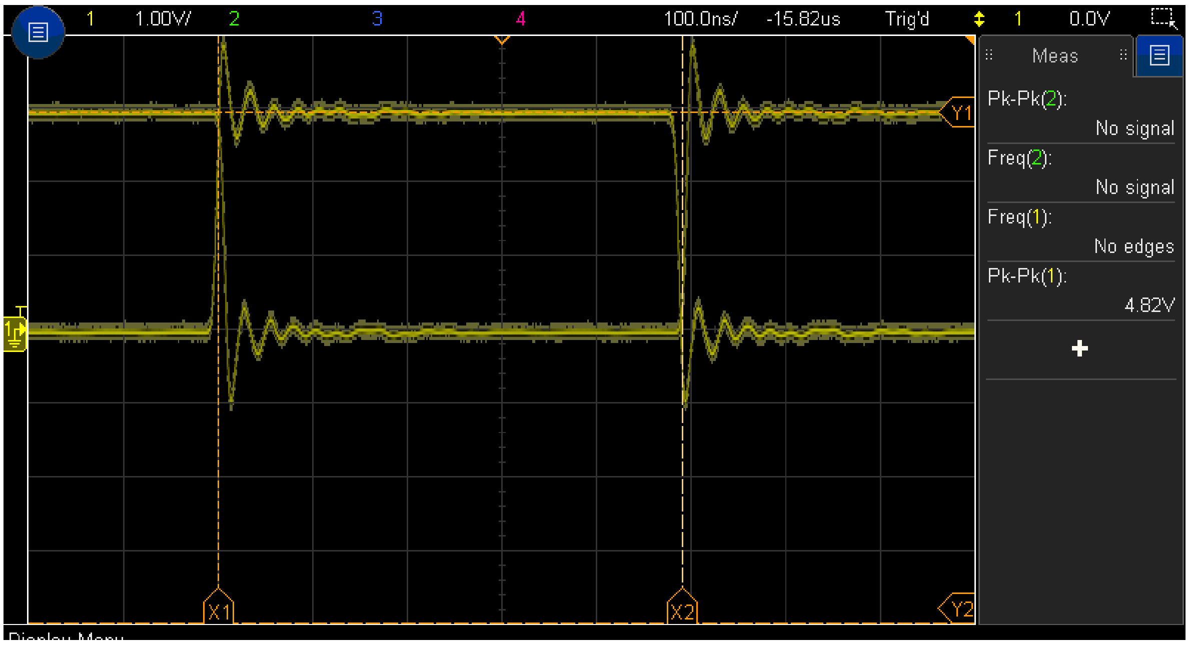


Figure 3.10: Eye pattern of the clock generated from function generator

is received as an interrupt at the MCU, the *DRDY* pin of the sensor is driven high inside the interrupt handler. Since this is a critical section of the firmware, the interrupt handler is kept as short as possible to reduce interrupt latency. The frequency of this synchronization pulse can also be varied according to the worst case synchronization error between any two sensors recorded for each case.

3.5.2. Subsequent Synchronization

In the implementation of the subsequent synchronization, periodic synchronization signals are not required. When the sink transmits a beacon to indicate the start of measurement to all the nodes, the nodes record this interrupt trigger according to its local clock and start measuring. In order to perform subsequent synchronization, the local time at every reception of the interrupt from the *INT1* pin of the sensor is recorded. The time stamps attached to the samples correspond to the own local MCU clock. The timestamp is given by a timer derived from a 48 MHz clock which has a 21 ns resolution approximately. The clocks need not be synchronized continuously. The memory stores the timestamped samples for later retrieval. Hence even in case of node failure, the memory will hold the last 30 seconds worth of information. The sample synchronization can be done after we retrieve the timestamps of the measurements at the sink/host. The time-shift between the sensor data is estimated and the data is re-sampled accordingly by compensating for the positive/ negative lag. The interpolation and re-sampling of the received data is done on the host. MATLAB was chosen as a favorable platform to perform such post-processing techniques.

4 Test Results and Analysis

4.1. Overview

This chapter describes the experimental setup and results from the implementation of synchronization approach discussed in Section 3.5. The testing method and specifications are explained in the forthcoming subsections. The observations recorded from performing a series of preliminary tests are also included. The implementation and the results of the real-time and subsequent synchronization techniques are elaborated in Section 4.3.

4.2. Testing Methodology and Specification

The quality of signals obtained from the sensor is an important factor when being used for high-precision applications such as experimental modal analysis. A basic sanity check for an accelerometer is to perform a gravitational orientation test by placing it flat on a fairly vibration-free surface and aligning the axis of accelerometer along the axis of rotation. This is described in Appendix A. One of the methods to test the reliability of an inertial measurement unit is to provide dynamic controllable excitation and observe the response of the sensor. A mechanical shaker is used to provide the input excitation signals. A shaker is an electro-dynamic exciter that delivers force and vibrates according to the input signal. This in turn excites any object such as an inertial sensor placed on a shaker. The data obtained from these tests are used to obtain the frequency response which is further used in modal analysis. A set of precursory tests to observe and analyze the behavior of the sensor and software were also performed which are discussed in Section 4.2.3.

4.2.1. Test Setup

In order to measure the controlled input, shaker tests were performed. To obtain a better understanding of the MEMS sensor performance, it was compared to a well-performing high quality reference sensor (Section 4.2.2). Figure 4.1 illustrates the test setup used. The sensor nodes and reference sensor were placed on a flat fixture atop the shaker. The shaker was driven by the signal generator which provided the various input excitation signals. Care was taken when mounting the sensors on the shaker platform and orienting them along the same axes. Since the shaker requires enough energy to excite, a compatible power amplifier is used. The actual testing equipment and the sensor nodes are shown in the Figure 4.2.

4.2.2. Reference Sensor

The reference sensor used for testing is the industrial standard piezoelectric tri-axial accelerometer PCB356B18 [27]. Piezoelectric sensors are extremely stable with high sensitivity. This sensor measures

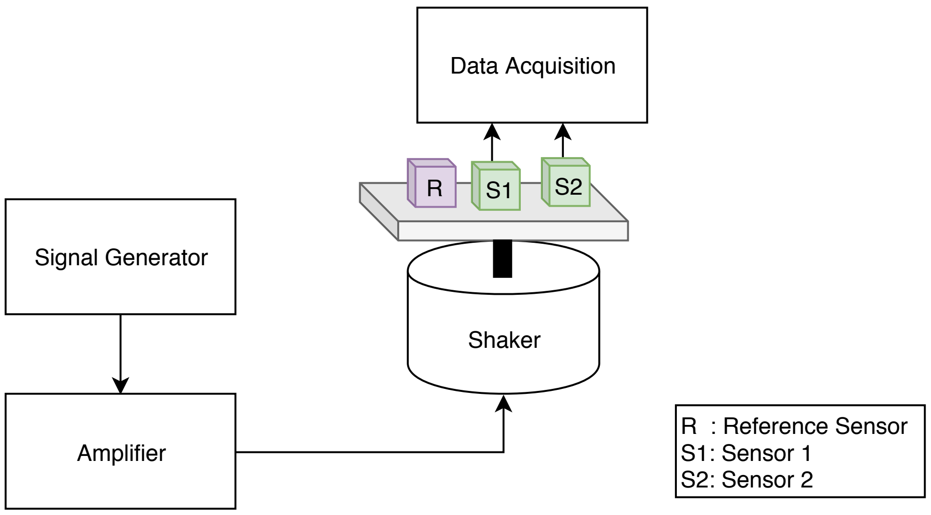


Figure 4.1: Test Setup

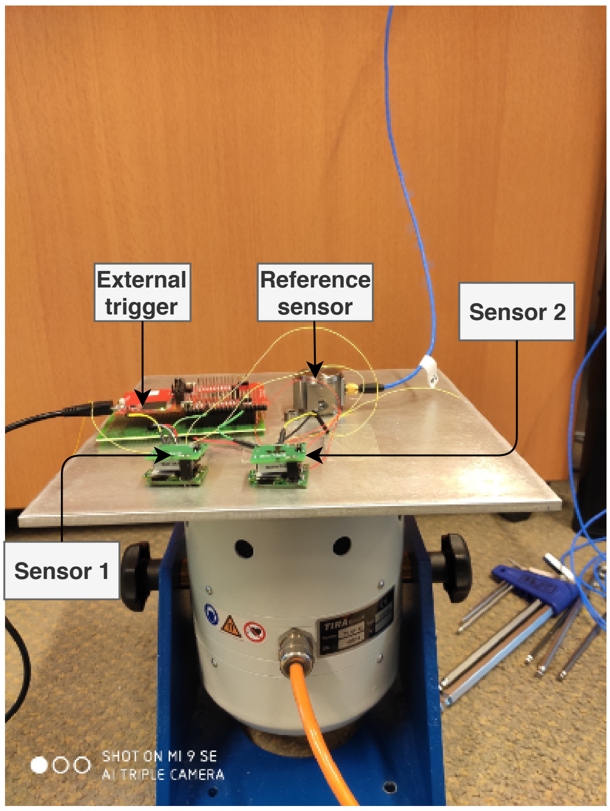
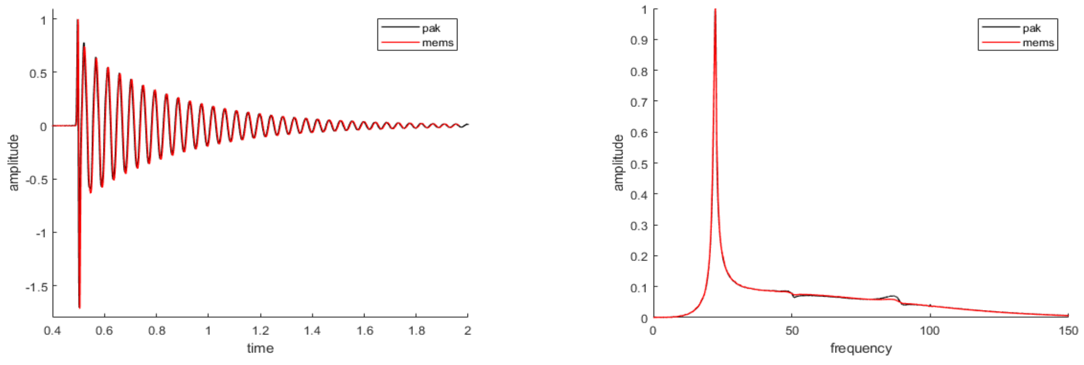


Figure 4.2: Sensor nodes and reference sensor on the shaker

upto $\pm 5g$. The time response of the reference sensor was compared with the decaying sine input signal from the function generator to cross-check the excitation of the mechanical shaker. The sensor nodes were compared in the time and frequency domains to that of the reference sensor. The corresponding plots are presented in Figure 4.3. The sampling frequency of the reference sensor was at 16 KHz. The data from the reference were acquired by the *pak* data acquisition system.



(a) Time-domain (b) Frequency-domain

Figure 4.3: Comparison of reference sensor(pak) and MEMS sensor for a decaying sine input

4.2.3. Preliminary testing and observations

In order to understand the behavior of the sensor and the impact of firmware on the performance, a set of preliminary tests were carried out. These included checking for variation in the sampling rates, sensor noise, effect of synchronization on the data, and impact of time delay on synchronization. The observations from these tests helped in gaining understanding of the system.

Sensor Bias and Noise

Under free-fall conditions, the acceleration value should ideally be zero. Zero-g offset is any deviation from this ideal value for free-fall. Any two devices of the same part number might differ in their zero-g offset which are influenced by their manufacturing and packaging process. The average signal output has a small offset even when the sensor is at rest which is also called bias of sensor. Since the biases change over time, the error in inertial measurement data also varies. The following measurement was done in a temperature controlled clean room with very litle external vibration. The sensor axes were oriented according to Figure A.1 each time to obtain the zero-g offset for each axis. Graph 4.4 shows the acceleration measurement for the positive z-axis pointing upwards along with a constant fit line. The constant fit line represents the mean shift of acceleration distribution for each axis [28]. The constant fit values for the acceleration distribution of six possible orientations are tabulated in 4.1. The positive and negative signs for the axes is to indicate that the axes were pointing upwards and downwards respectively. The sensor bias values are calculated from the fit values and were subtracted from the actual measurements to obtain true acceleration data. This is represented by :

$$A_{true}(g) = (A_{actual} - A_{bias})/Gain \tag{4.1}$$

where A_{actual} and A_{bias} are in terms of g, and $Gain$ is assumed to be unity for simplicity. However, the misalignment in angle tilt is also to be considered while calculating the gain. The sensor bias values for each axis is shown in Table 4.2.

From a similar measurement environment, the noise level of the sensor was also obtained. The standard deviation of the value from the constant fit provided the noise component of the signal. The theoretical noise value was calculated in Section 2.6.2 to be 3.3 mg. Table 4.3 gives the average standard deviation of noise for a certain number of measurement runs. It can be inferred that the noise level complies with that of the calculated theoretical value. As mentioned earlier, the noise places a limit on the resolution of the acceleration detection.

Sampling Rate Variation

The requirement is to sample the acceleration data at 1 KHz. A simple test was conducted to test if there were any variations in the number of samples collected. Table 4.4 shows the number of samples collected under three different conditions:

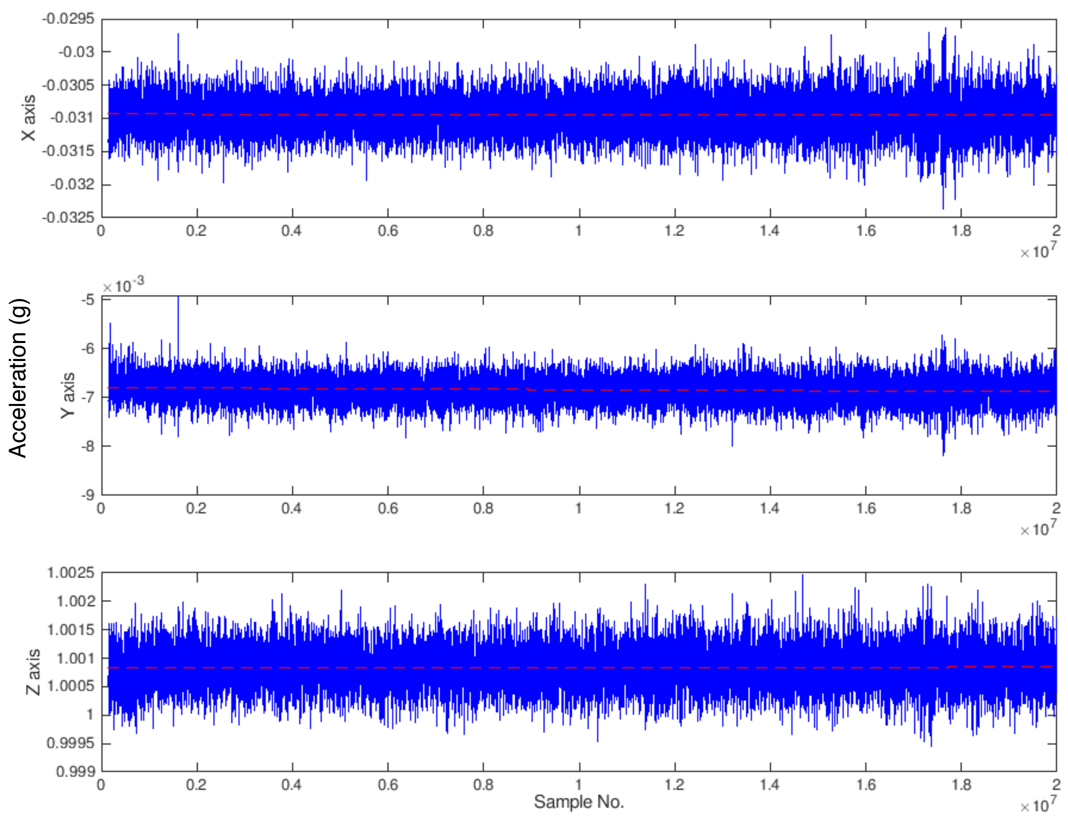


Figure 4.4: Acceleration data for positive z-axis orientation with the constant fit line

Table 4.1: Constant fit value for the different sensor orientations

Orientation	**x-axis** $(10^{-3}g)$	**y-axis** $(10^{-3}g)$	**z-axis** $(10^{-3}g)$
+X	1027.4	18.9	-16.4
-X	-1033.1	22.3	-12.1
+Y	-16.9	1034	10.3
-Y	-12.5	-1021	-19.2
+Z	-31.0	-7.1	1001
-Z	-21.3	-10.5	-1012

Table 4.2: Sensor bias for the three axes

Axis	**Bias** $(10^{-3}g)$
x	-14.566
y	6.100
z	-8.066

Table 4.3: Standard deviation of acceleration data for five test sets

Test No.	**x-axis** $(10^{-3}g)$	**y-axis** $(10^{-3}g)$	**z-axis** $(10^{-3}g)$
1	0.26730	0.25585	0.34562
2	0.25598	0.25598	0.34478
3	0.28230	0.26743	0.35548
4	0.27061	0.25737	0.34474
5	0.26230	0.25950	0.34537

With Ext Sync (FG) External synchronization for the sensor with a clock source from the function generator at ~0.2 ppm

With Ext Sync (GPT) External synchronization for the sensor with a clock source from the general purpose timer clock at ~1600 ppm

Without Ext Sync Sensor with its own internal clock source without external synchronization.

Table 4.4: Sampling rate comparison

Conditions	Samples/1 sec	Samples/30 sec	Actual sampling rate (Hz)
With Ext Sync (FG)	1000	30000	1000
With Ext Sync (GPT)	1002	30060	1002
Without Ext Sync	985	29517	983.9

The number of samples for one second and 30 seconds are listed along with the corresponding sampling in Table 4.4. If the sensor works under its own internal clock, it collects lesser samples than the estimated number. In case of external synchronization, two different clock sources are provided and the corresponding variation in the number of samples was recorded. It can be understood that the clocks are one of the main sources of variation in sampling frequency. The better the clocks, the less the sampling rate varies.

Sample Corruption

Even though the use of external synchronization for the sensors provides better stability in terms of sampling rate, there is a drawback that was observed. For every synchronization trigger pulse given to the sensor, the two subsequent samples were corrupted. This can be seen in Figure 4.5, where the vertical lines denotes the corrupted data points. A 10 Hz sine signal was given as an input to the mechanical shaker on which the sensor was placed. A synchronization pulse was given at a frequency of 0.1 Hz (period = 10 sec) for a measurement interval of 40 seconds. The sample corruption was clearly visible in the plot after the synchronization pulse. This is due to the resetting of the decimation filters after the pulse. The output data rate filters realigned with respect to the previously received synchronization pulse. This caused the samples to be re-sampled incorrectly by the ADCs. Hence the samples after the synchronization pulse read the extreme range of the accelerometer. Since the range was set at $\pm 2g$, the corrupted samples read around this value.

Interrupt Propagation Delay

For the synchronization to work as proposed, the external synchronization pulse need to be registered in the microcontroller and the subsequent signal should be fed to the sensor with minimal propagation delay. This critical function was hence kept in an interrupt routine to ensure that the system responds quickly to external triggers. The external trigger was generated from the function generator at the required frequency. It was set at 1 Hz to emulate the reception of beacon packets from the sink node via BLE. Figure 4.6 shows that there is a clear propagation delay when the synchronization signal is generated and its reception at each sensor. Table 4.5 summarizes this delay averaged over a period of 40 seconds for three test runs. A zoomed-in Figure 4.7 shows that there is a difference in the reception time for each of the sensors. Even though it is in the range of few microseconds, it is useful to know this information for further optimization. The values are provided in Table 4.6.

Table 4.5: Delay in registering synchronization interrupt

Min (ns)	**Max** (ns)	**Mean** (ns)	**Standard deviation** (ns)
963	1000	984.44	9.69

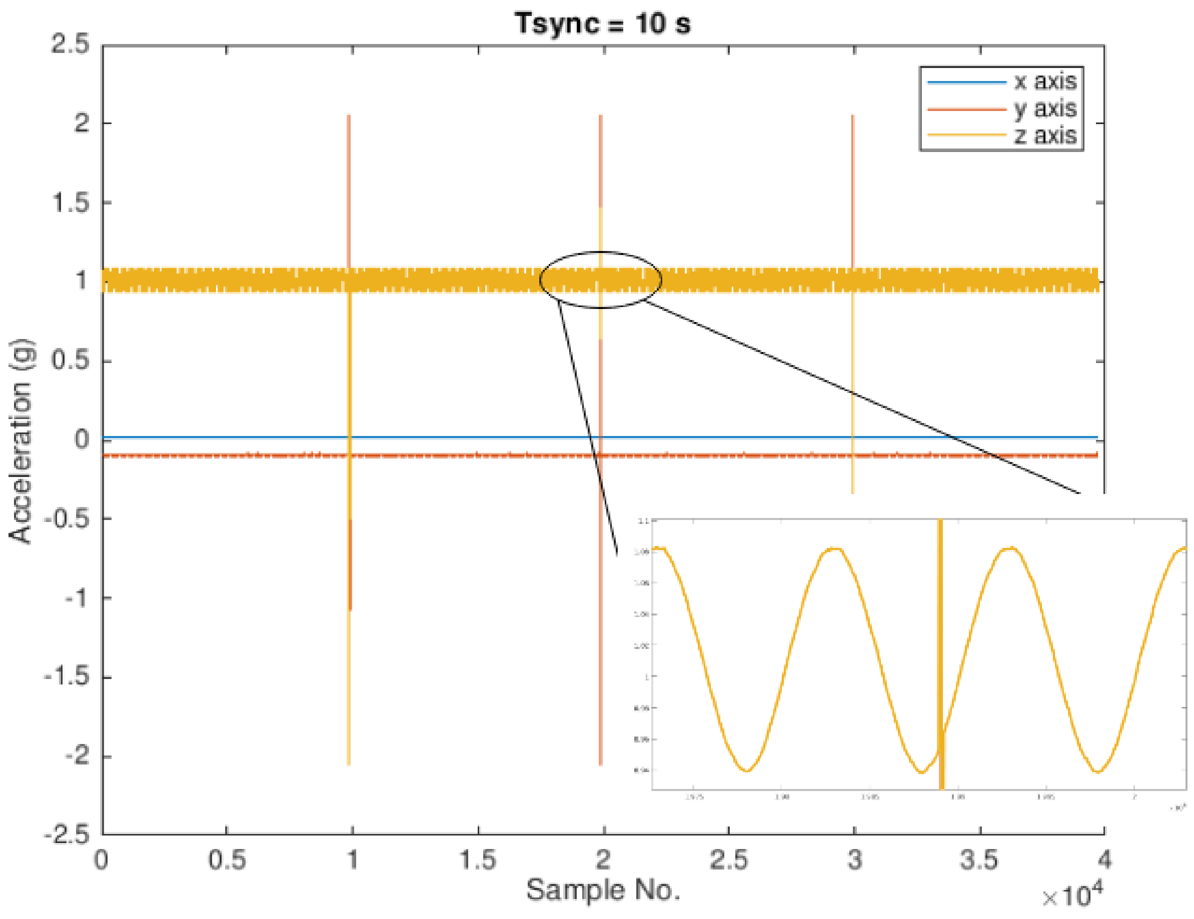


Figure 4.5: 10 Hz sine input to sensor for 40 seconds depicting sample corruption after every sync pulse

4.3. Synchronization Tests and Results

The objective was to acquire synchronous samples by aligning their sampling instants across the sensor nodes. It can be inferred from the previous tests that the master clock source for the sensor plays an important role in enabling the synchronization at the sampling level. In order to substantiate the inference, two sensor nodes were fed with the same clock source. The result was that the ADC sampled exactly at the same instant for all the sensors. This can be seen in Figure 4.8.

The frequency stability of the clock in this case does not matter, since the synchronization error is due to the relative drift of the respective sensor clocks. The following subsections elaborate on the obtained synchronization error for the two methods of synchronization. The tests were done to emulate the practical application as close as possible.

Table 4.6: Interrupt propagation delay between 3 sensor nodes

	Min (ns)	**Max (ns)**	**Mean (ns)**	**Standard Deviation (ns)**
Sensor 1 & 2	1.0	12.1	5.42	3.32
Sensor 1 & 3	1.0	13.0	6.64	3.85
Sensor 2 & 3	1.7	10.4	5.08	2.92

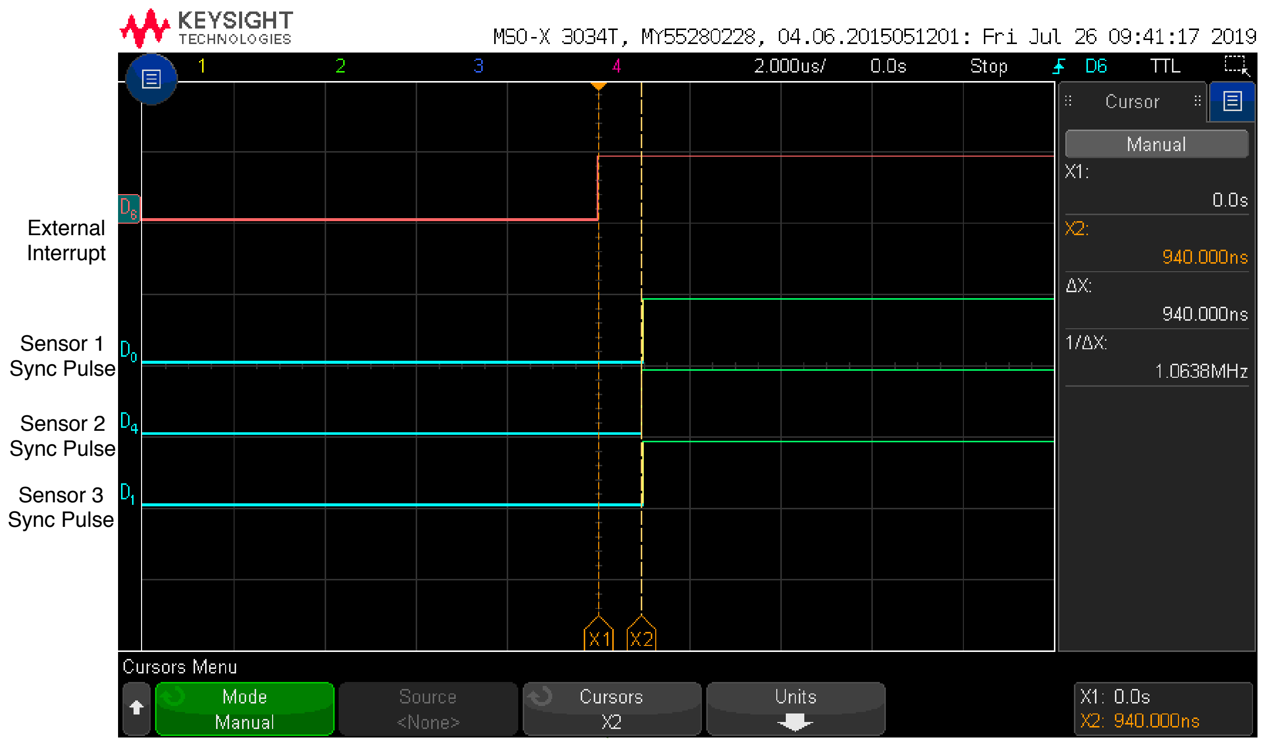


Figure 4.6: Propagation delay of external interrupt

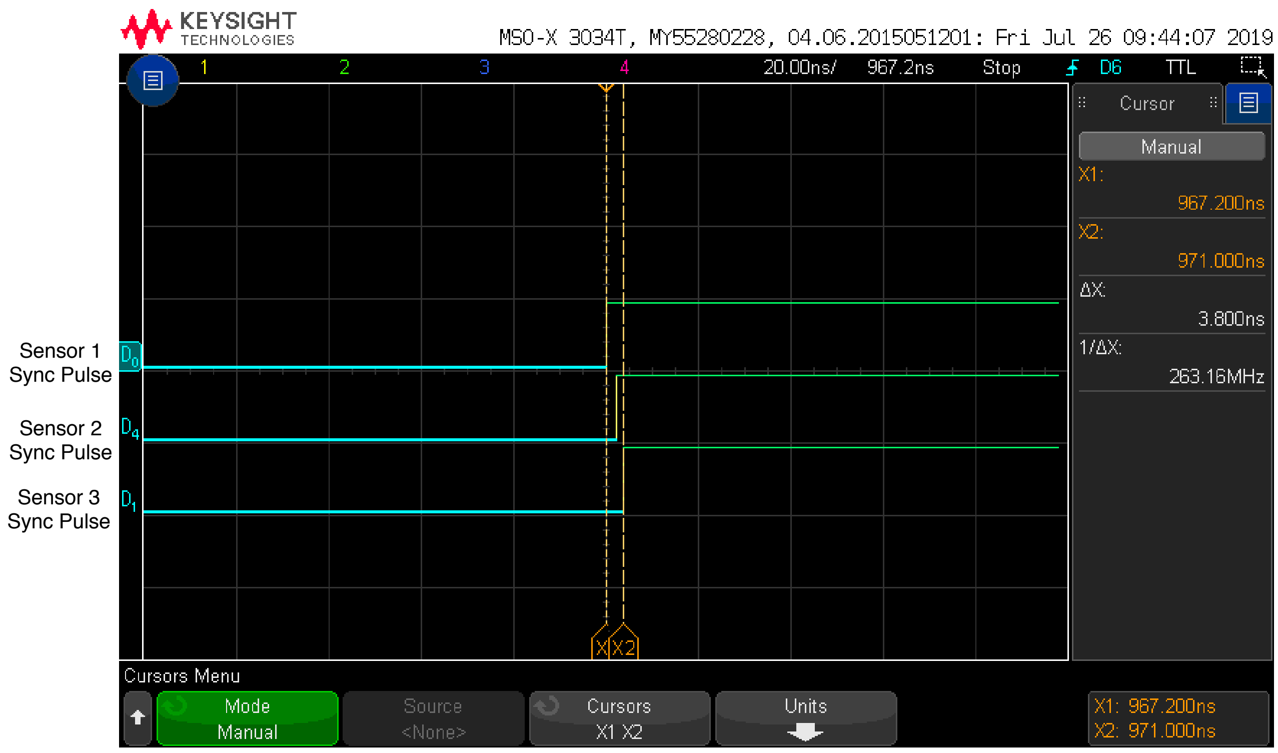


Figure 4.7: Interrupt propagation delay between three sensor nodes for sync pulse

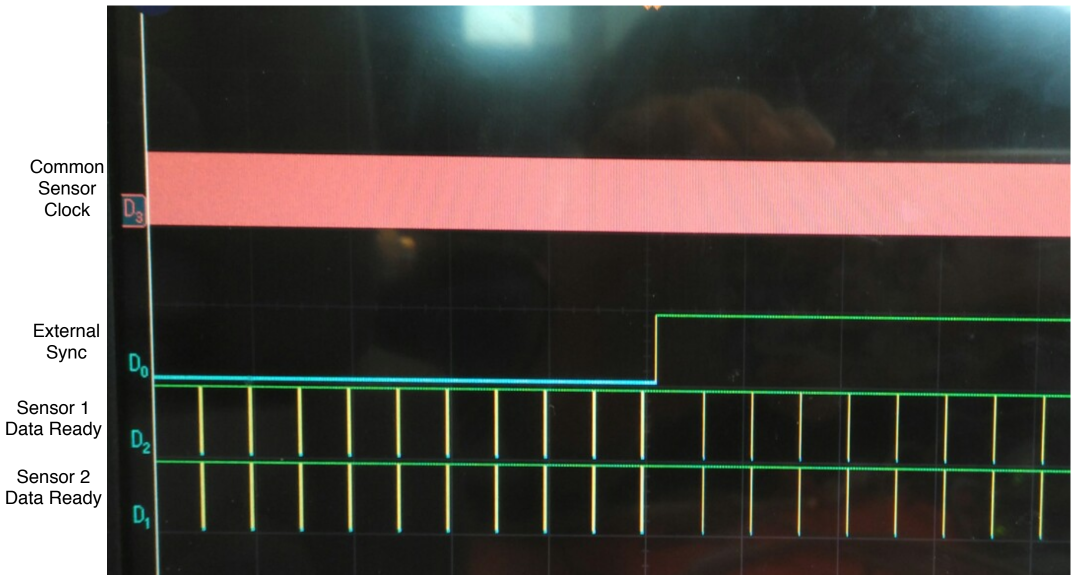


Figure 4.8: Common clock and synchronization interrupt given to sensor nodes

4.3.1. Real-Time Synchronization

The synchronization tolerance for the real-time technique was analyzed for both clock sources. Figure 4.9 shows how the sampling instant aligns after the synchronization interrupt. This is for the re-synchronization period of 1 second for the three sensor nodes. The sensors were fed with general purpose timer clocks. The worst case time difference between the sampling instants are around 500 μs. The frequency stability of the sensor clock and synchronization accuracy are related by:

$$T_{sync} = \frac{\Delta t_{sync\_acc}}{ppm} \tag{4.2}$$

where T_{sync} is the re-synchronization period required to achieve the synchronization accuracy, Δt_{sync_acc} for a clock with a frequency stability in ppm.

Substituting the frequency accuracy of the general purpose timer with 1600 ppm and the required accuracy of synchronization with 55.55 μs in Equation 4.2, results in re-synchronization period of 30 ms. The sensor with a clock source from the general purpose timer and a re-synchronization period of 32ms is shown in Figure 4.10 for two sensors. Figure 4.10 shows the plot for synchronization error between two sensor nodes for a re-synchronization period of 32ms. The plots depict the experimental data obtained from the logic analyzer by scoping the required signals from the sensors and external trigger. We can see that after the synchronization pulse is given, the error reduces to a small value (<5 μs) and again gradually rises to around 58 μs. A zoomed-in capture of the scope is given in Figure 4.11.

As shown in Table 3.1, the frequency stability of the function generator is almost ideal. However, in a real application scenario, crystals are never accurate and tend to drift over time, temperature, humidity, external vibration, etc. Hence to emulate the behavior of a market-available crystal oscillator to be used as a sensor clock, the relative frequency variation of the function generator was kept around 40 ppm and 50 ppm. This was done by proving a 40 Hz and a 50 Hz relative difference respectively between the two function generators as shown in Table 4.7.

With the function generator emulating the drifting behavior of a practical crystal oscillator, the synchronization test was performed for a re-synchronization period of 1 second. The expected synchronization error between the sampling times of the two sensors was 40 μs and 50 μs respectively.

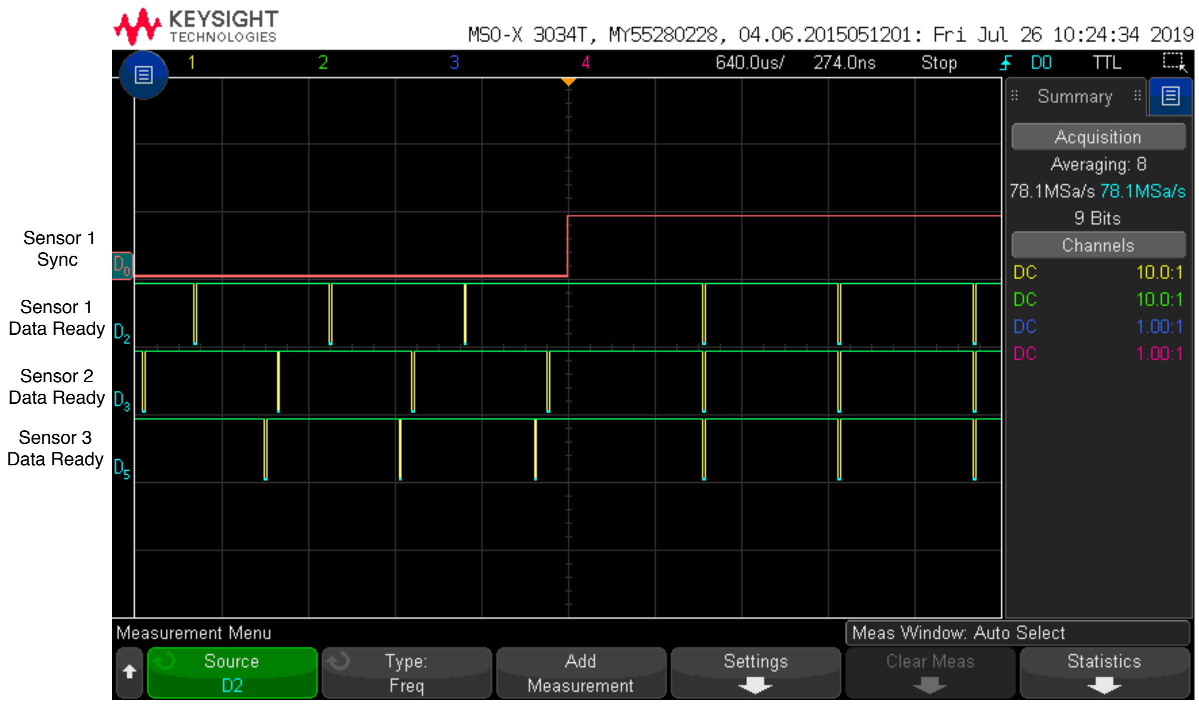


Figure 4.9: Data Ready Interrupts synchronizing after the synchronization pulse

Table 4.7: Function generator frequency settings

Relative frequency accuracy (ppm)	**FG 1 frequency** (MHz)	**FG 2 frequency** (MHz)
40	1.024000	1.024040
50	1.024000	1.024050

The corresponding synchronization error for the two frequency accuracy setting are given in Figures 4.12 and 4.13 respectively. The captures from the logic analyzer is also given in Figures 4.14 and 4.15.

4.3.2. Subsequent Synchronization

As described in Section 3.5, the subsequent synchronization is done as a post-processing technique. The timestamps of the samples assigned according to the local microcontroller clock were used to estimate the time-shift and align the samples. In order to enable synchronization, the time-stamps were added in the interrupt handler right after the ADC interrupt was registered. MATLAB was used for processing the obtained acceleration data and its corresponding timestamps from each sensor.

The data plot for the two sensors for a decaying sine input signal is given in Figure 4.16. The time-shift was measured to be 0.005 seconds. Post-processing this data involved estimating the time shift and re-sampling one sensor with respect to the other in order to align them in time to achieve synchronization. The offset in the amplitude of the signals was normalized to zero by subtracting the mean of the signal from each sample point. The index corresponding to the maximum correlation was used to obtain the lag between the signals. Figure 4.17 shows the time-shifted and re-aligned signals from the two sensors.

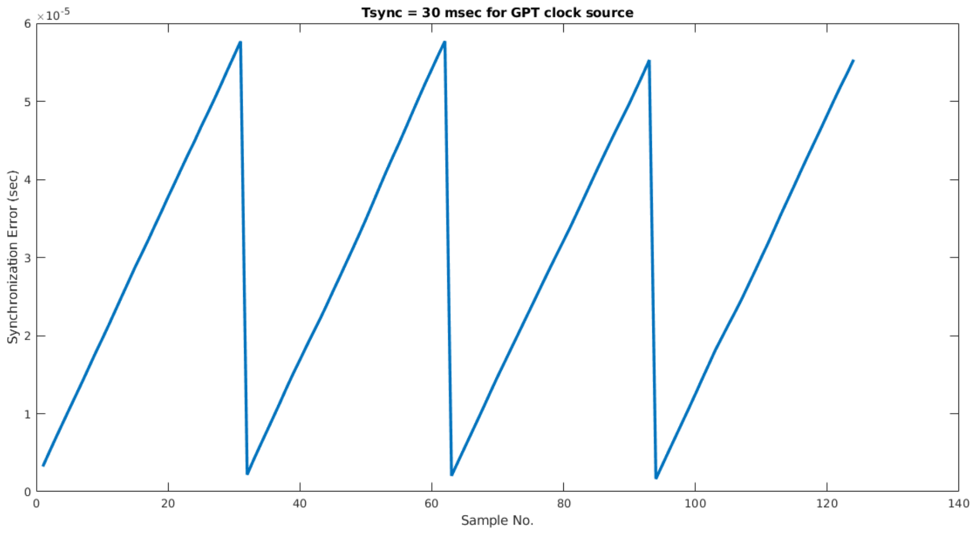
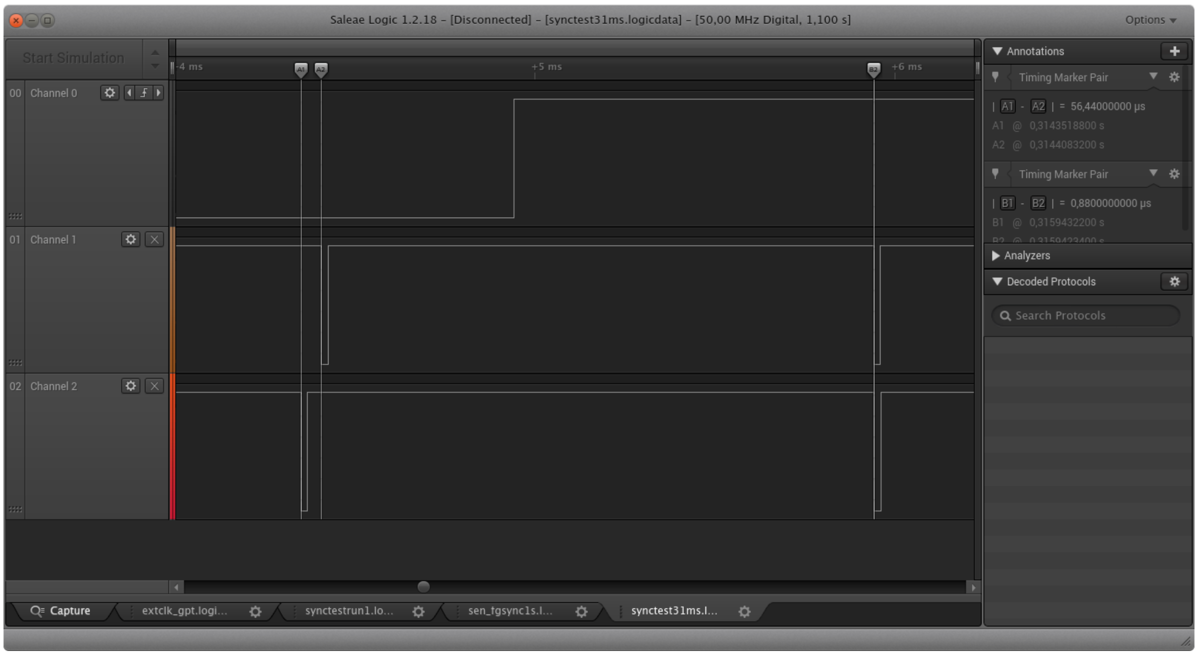


Figure 4.10: Synchronization tolerance for real-time synchronization


Saleae Logic 1.2.18 - [Disconnected] - [synctest31ms.logicdata] - [50,00 MHz Digital, 1,100 s]

Timing Marker Pair
| A1 - A2 | = 56,44000000 µs
A1 @ 0,3143518800 s
A2 @ 0,3144083200 s

Timing Marker Pair
| B1 - B2 | = 0,8800000000 µs
B1 @ 0,3159432200 s
B2 @ 0,3159423400 s

Figure 4.11: Capture of the scope showing the worst and best case error for Plot 4.10

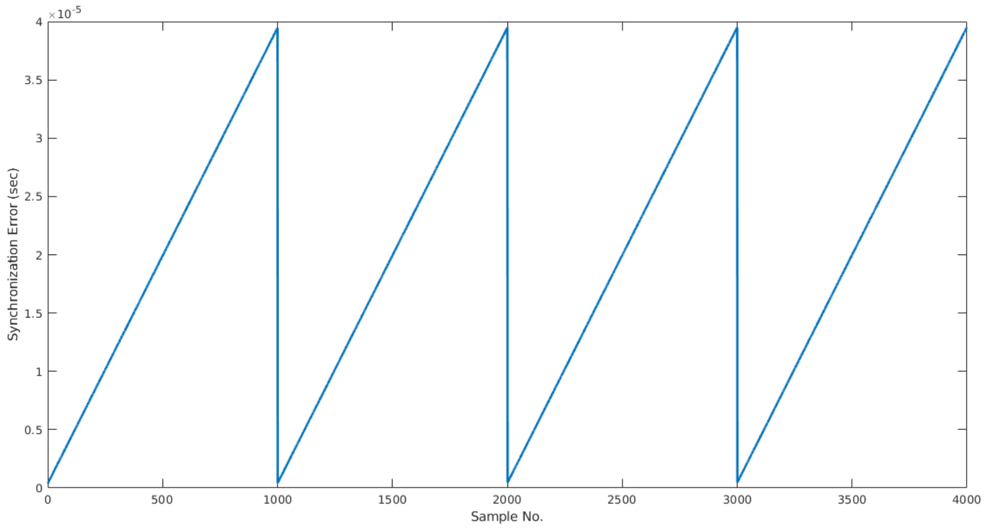


Figure 4.12: Synchronization tolerance for a clock source with 40 ppm relative frequency error

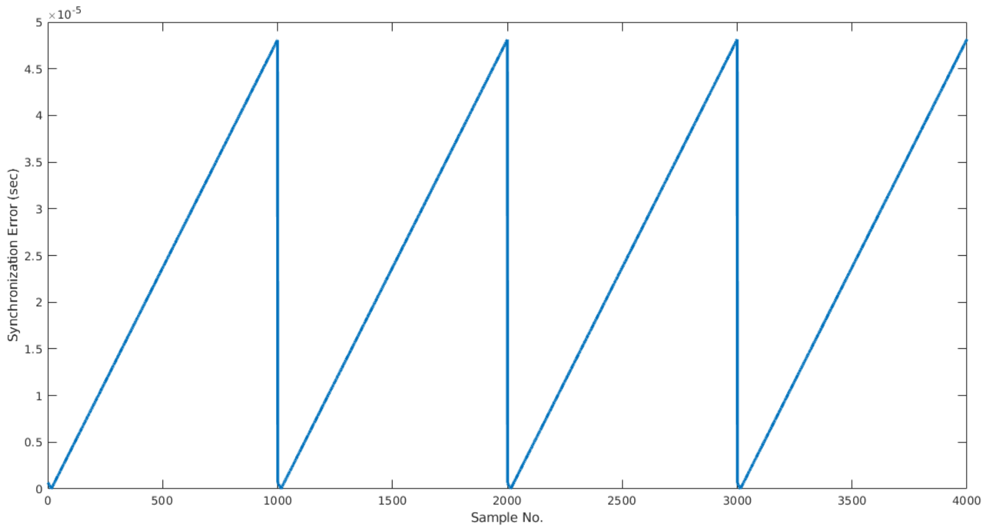


Figure 4.13: Synchronization tolerance for clock source with 50 ppm relative frequency error

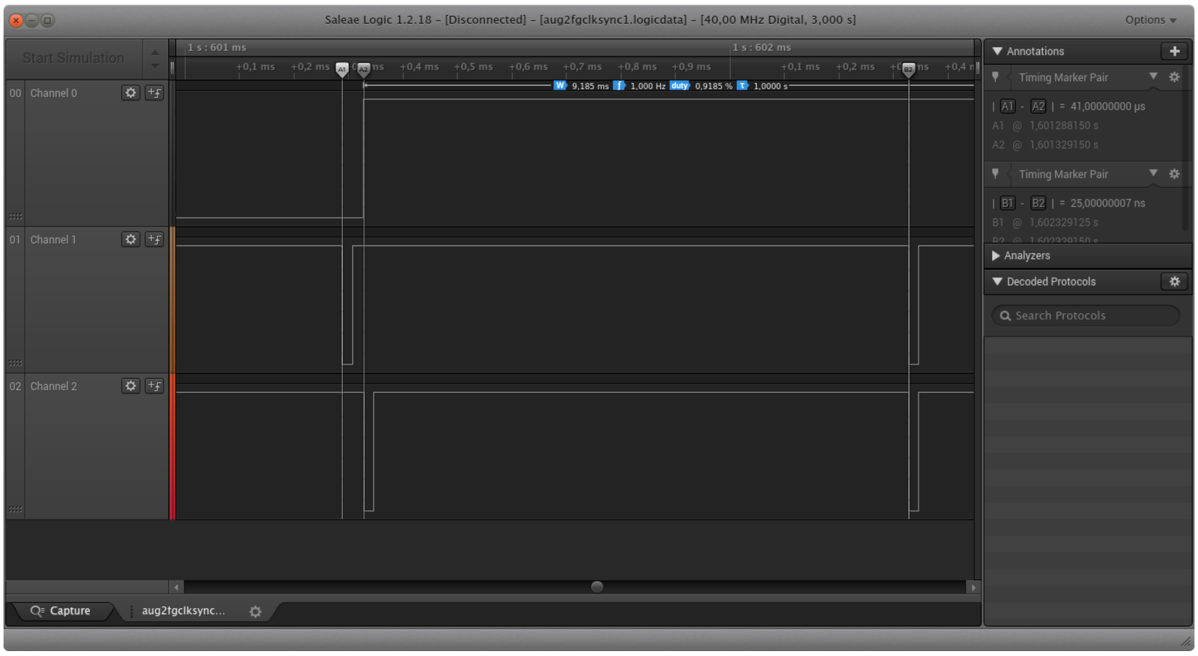


Figure 4.14: Capture of the scope showing the worst and best case error for Plot 4.12

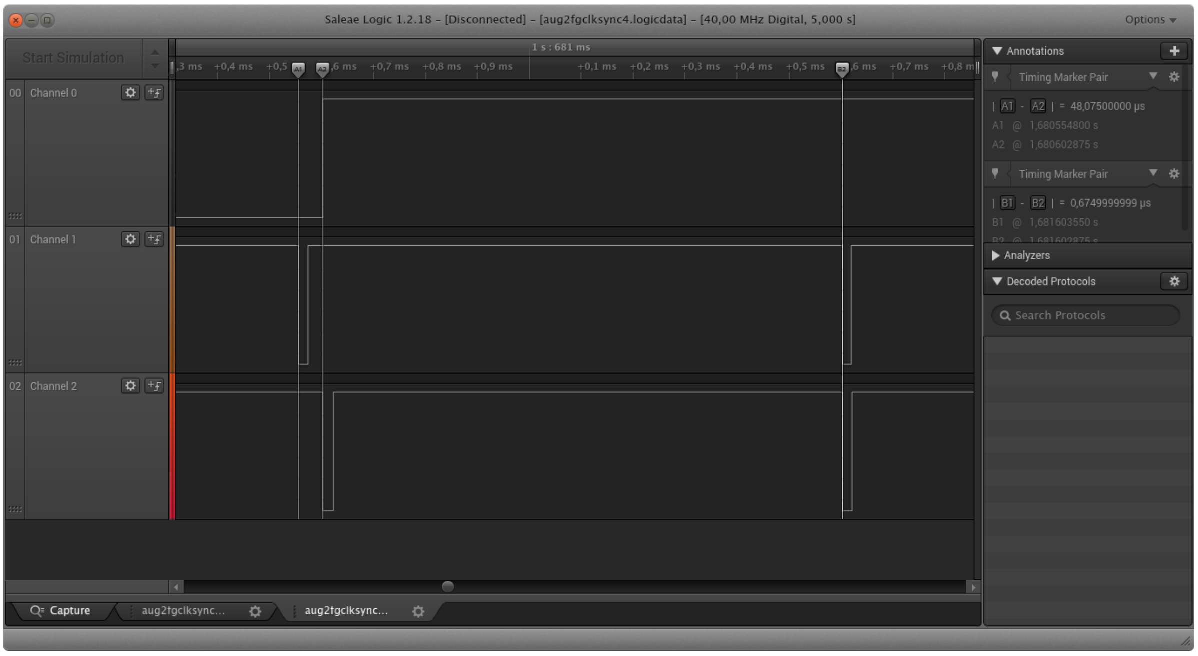


Figure 4.15: Capture of the scope showing the worst and best case error for Plot 4.13

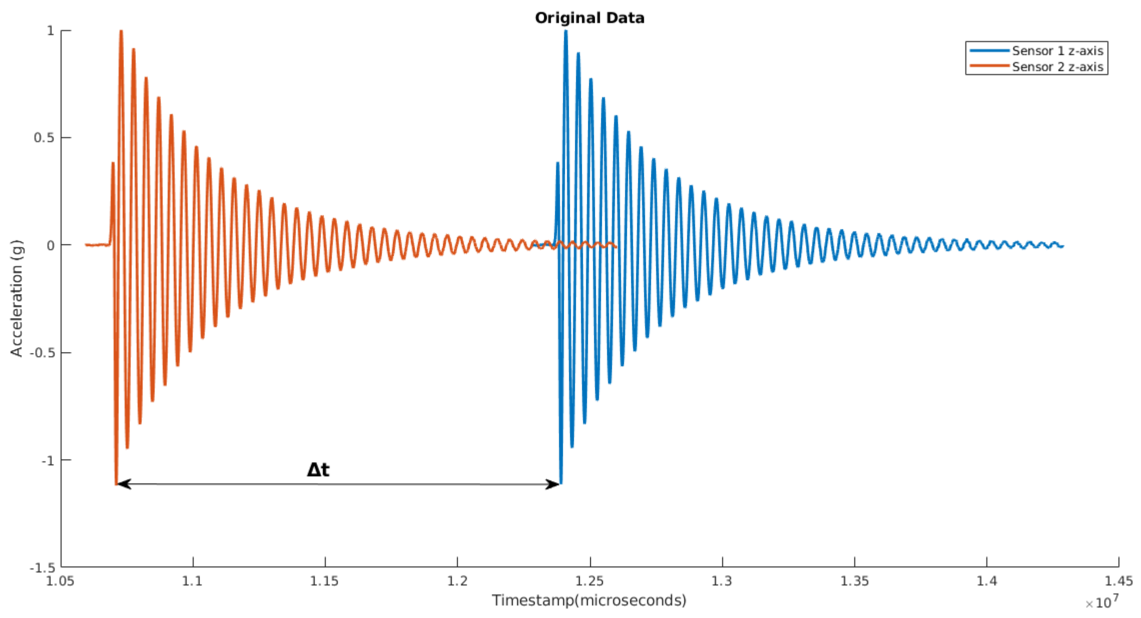


Figure 4.16: Acceleration data plot for two sensors capturing the same input signal **before** subsequent synchronization

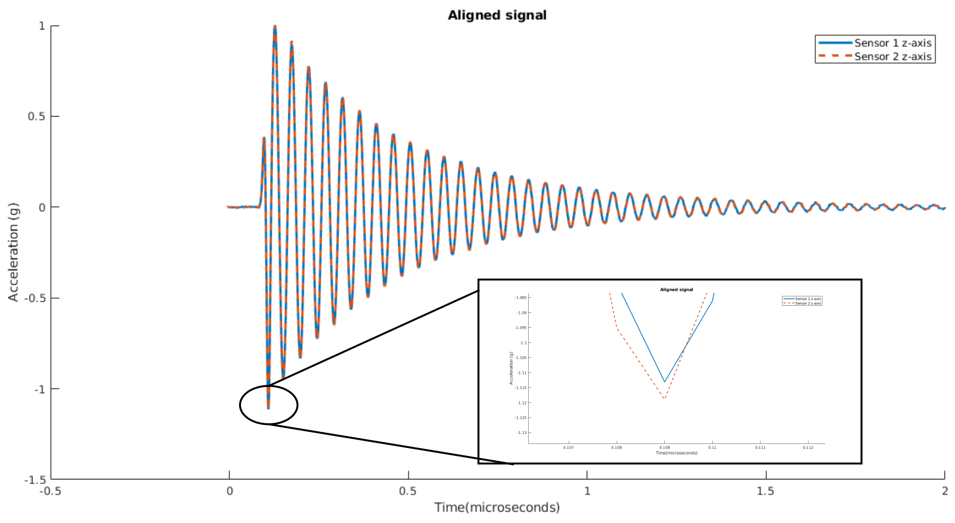


Figure 4.17: Acceleration data plot for two sensors capturing the same input signal **after** subsequent synchronization

5 Conclusion

Use of wireless networks for monitoring purposes has been deemed advantageous over wired networks. It can be used to remotely obtain a quantified perception of the application environment. The proposed distributed network is to analyze the dynamics of the wafer handler robots inside the photolithographic machine at ASML. In order to obtain reliable information, accelerometer sensors must be placed at various locations inside the machine and obtain samples. The requirement for successful sensor fusion is that the samples must be obtained with high synchronization.

In this thesis, the emphasis was placed on obtaining synchronous samples across sensors in a network. The importance and effects of synchronous sampling was discussed. Two approaches to achieve synchronization in samples were introduced and implemented. The real-time synchronization uses a software solution by providing interrupt-based triggering which realigns the ADC operation of the sensor. It was shown that with the use of reasonably stable clocks for the sensors, a synchronization tolerance of the required 55 μs can be achieved in the real-time synchronization technique. The 55μs mainly comprises of the network synchronization for beacon transmission and the sampling time variation between any two sensors in the network. The trade-off here is that there is data corruption after every call for synchronization. The subsequent synchronization approach is based on post-processing the obtained signals along with their timestamps to align them in time by re-sampling. The drawback, however, is that the signals are approximated due to interpolation and re-sampling. The use of timestamps further increases the overhead on the packet payload to be stored in memory and transmitted to the sink. Hence this provides a relatively easy solution to synchronous sampling, but at the cost of data approximation/quantization error. If the application does not allow room for such errors, then the real-time technique is the most suitable. With a stable clock compensating wireless network infrastructure, the real-time synchronization technique stands as a proof of concept.

5.1. Recommendations for Future Work

This work has laid the foundation for further optimization. A few possible directions for future work are listed below.

- An interface is to be built between the network and sensor to enable the wireless transfer of acceleration data from the memory to the sink node.
- Redesigning of the sensor node PCB is required to include the proposed crystal oscillator as a clock source for the sensor.
- Vacuum compatible housing is required for the sensor node to be placed in the vacuum wafer stage of the photo-lithography machine.

Bibliography

[1] U. Uyumaz and Technische Universiteit Eindhoven (TUE). Stan Ackermans Instituut. Software Technology (ST), *Wafer flow simulator visualizer*, Ph.D. thesis (2013), eindverslag.

[2] K. F. Kiefer, B. Swanson, E. Krug, G. Ajupova, and P. L. Walter, *Wireless Sensors Applied to Modal Analysis,* .

[3] V. Krishnamurthy, K. Fowler, and E. Sazonov, *The effect of time synchronization of wireless sensors on the modal analysis of structures,* Smart Materials and Structures **17** (2008), 10.1088/0964-1726/17/5/055018.

[4] A. K. Andreas Engel, *Demo: The need for wireless clock drift estimation and its acceleration on a heterogeneous sensor node,* EEE Proc. Conference on Local Computer Networks (LCN), Clearwater Beach, Florida (USA) (2015).

[5] B. Bengherbia, M. O. Zmirli, A. Toubal, and A. Guessoum, *FPGA-based wireless sensor nodes for vibration monitoring system and fault diagnosis,* Measurement **101**, 81 (2017).

[6] B. Jiang, M. Chen, and F. Chen, *A clock drift compensation method for synchronous sampling in sensor networks,* Measurement Science and Technology **30** (2019), 10.1088/1361-6501/aaf6c7.

[7] J. Elson, L. Girod, and D. Estrin, *Fine-Grained Network Time Synchronization using Reference Broadcasts **, Tech. Rep. (2002).

[8] M. L. Sichitiu and C. Veerarittiphan, *Simple, Accurate Time Synchronization for Wireless Sensor Networks*, Tech. Rep.

[9] K. Römer, P. Blum, and L. Meier, *Time Synchronization and Calibration in Wireless Sensor Networks,* in *Handbook of Sensor Networks* (John Wiley & Sons, Inc., 2005) pp. 199–237.

[10] J. Funck and C. Gühmann, *Comparison of approaches to time-synchronous sampling in wireless sensor networks,* Measurement: Journal of the International Measurement Confederation **56**, 203 (2014).

[11] Z. Feng and L. S. Katafygiotis, *THE EFFECT OF NON-SYNCHRONOUS SENSING IN WIRELESS SENSORS ON STRUCTURAL MODAL IDENTIFICATION*, Tech. Rep.

[12] M. A. Javaid, *Wireless Sensor Networks: Software Architecture,* (2014).

[13] C. Shore and T. Manager, *Efficient Interrupts on Cortex-M Microcontrollers*, Tech. Rep.

[14] A. T. D. Zeo, *A wireless sensor network for machine dynamics performance monitoring,* (2019).

[15] A. Devices, *ADXL354/ADXL355 (Rev. A),* (2016).

[16] Cypress, *CY15B104Q FRAM 4Mbit Memory,* (2017).

[17] Texas Instruments, *CC2640R2F BLE Microcontroller,* (2017).

[18] K. N. N. Yazdi, F. Ayazi, *Micromachined inertial sensors,* Proceedings of the IEEE **86**, 1640 (1998).

[19] C. Li, R. Azzam, and T. Fernández-Steeger, *Kalman filters in geotechnical monitoring of ground subsidence using data from MEMS sensors,* Sensors **16**, 1109 (2016).

[20] M. Andrejašic, *Mems accelerometer - seminar,* (2008).

[21] A. Devices, *Fundamental Principles Behind the Sigma-Delta ADC Topology: Part 1,* (2016).

[22] McCarthy Mary, *Peak-to-Peak Resolution Versus Effective Resolution,* .

[23] A. Jerraya and W. Wolf, *Hardware/software interface codesign for embedded systems,* Computer **38**, 63 (2005).

[24] *Keysight Technologies 33500B Series Waveform Generators*, Tech. Rep.

[25] *Keysight 53131A/132A/181A Counters Data Sheet Recommended replacement products: 53200 Series RF and universal frequency counter/timers (Data sheet publication number: 5990-6283EN)*, Tech. Rep.

[26] K. Technologies, *InfiniiVision 3000T X-Series Oscilloscopes*, Tech. Rep.

[27] *Industrial standard piezoelectric tri-axial accelerometer pcb356b18,* (2007).

[28] J. Joemon and J. Mullassery, *EMBEDDED DATA ACQUISITION PLATFORM FOR LOW SPEED AUTOMATED SURFACE MEASUREMENTS ON ASPHALT*, Tech. Rep. (2015).

A
Sensor orientation

The data coming out of the sensor is raw acceleration. Hence, in order to get meaningful information from it, the data have to be combined and scaled accordingly. A sanity check to confirm the correct retrieval of information, is shown in figure A.1. The values adjacent to each orientation is the ideal acceleration output. The data plot in figure 3.5 is obtained by placing the accelerometer in the positive Z direction pointing upwards (green box in A.1). These orientations are also used to obtain the sensor bias and noise of the accelerometer sensor.

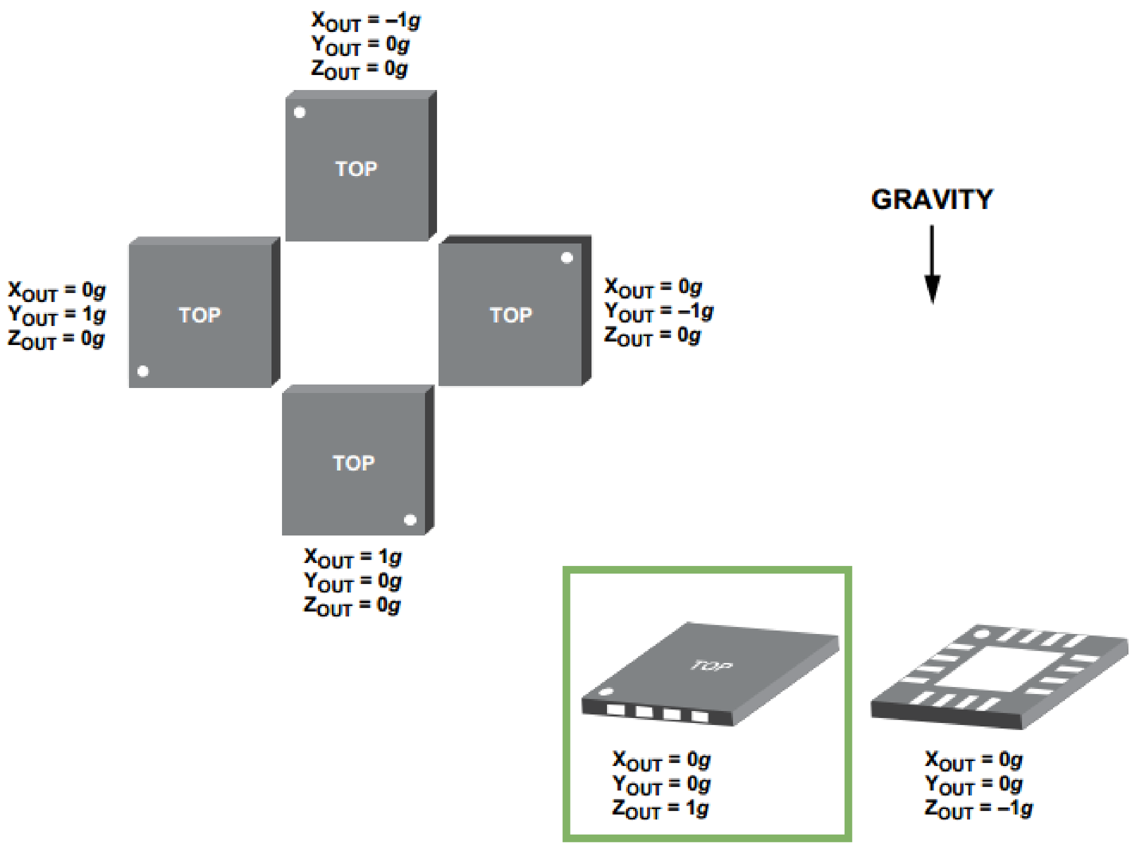


Figure A.1: Accelerometer output with varying orientation to gravity

B

General Purpose Timer clock generation :Listing

The code listing is of the function *Sen_HW_Clock_Setup* which generates the required 1.024 MHz clock from the general purpose timer of the microcontroller in the pulse width modulated mode. The generated signal is routed to the general purpose input/output pin 10 of the microcontroller which is further given as an input to the INT2 pin of the ADXL355 sensor. Since the signal is to be derived from the 48 MHz clock, the load value of timer is 46 to obtain a 1.024 MHz frequency.

```
static void Sen_HW_Clock_Setup(uint32_t timer_base)
{
    const uint32_t IOID = BRD_SEN_INT2;    //DIO_10 Pin on chip
    const uint32_t TIMER_LOAD_VAL = 46;    //48MHz/46 = ¬1.024MHz

    // Set configuration parameters according to the timer number
    uint32_t port_id = 0, subscriber = 0, event_source = 0, periph_timer = 0;
    if (timer_base == GPT0_BASE)
    {
        port_id = IOC_PORT_MCU_PORT_EVENT0;
        subscriber = EVENT_O_GPT0ACAPTSEL;
        event_source = EVENT_GPT0ACAPTSEL_EV_PORT_EVENT0;
        periph_timer = PRCM_PERIPH_TIMER0;
    }
    else if (timer_base == GPT2_BASE)
    {
        port_id = IOC_PORT_MCU_PORT_EVENT4;
        subscriber = EVENT_O_GPT2ACAPTSEL;
        event_source = EVENT_GPT2ACAPTSEL_EV_PORT_EVENT4;
        periph_timer = PRCM_PERIPH_TIMER2;
    }

    // Map timer event to GPIO and register CPU event
    IOCPortConfigureSet(IOID, port_id, IOC_STD_OUTPUT);
    EventRegister(subscriber, event_source);

    // Enable timer peripheral
    PRCMPeripheralRunEnable(periph_timer);
    PRCMLoadSet();
    while(!PRCMLoadGet());

    // Configure and enable timer according to steps in the datasheet
// 1. Ensure the timer is disabled (clear the TnEN bit) before making any changes.
```

```
    TimerDisable(timer_base, TIMER_BOTH);
// 2. Write the GPTM Configuration Register (GPT:CFG) with a value of 0x0000 0004.
    HWREG(timer_base + GPT_O_CFG) = 0x00000004;
// 3. In the GPTM Timer Mode Register (GPT:TnMR), write the TnCMR field to 0x1 and write…
     the TnMR field to 0x2.
    HWREG(timer_base + GPT_O_TAMR) |= 0b1010;
// 4. Configure the output state of the PWM signal (whether or not it is inverted) in …
    the GPTM Control Register (GPT:CTL) TnPWML field.
// 5. If a prescaler is to be used, write the prescale value to the GPTM Timer n …
    Prescale Register (GPT:TnPR).
//    TimerPrescaleSet(timer_base, TIMER_A, 255); // xxx
//    TimerPrescaleMatchSet(timer_base, TIMER_A, 0); // xxx
// 6. If PWM interrupts are used, configure the interrupt condition in the GPT:CTL …
    TnEVENT register field, and enable the interrupts by setting the GPT:TnMR TnPWMIE …
    register bit.
//  HWREG(timer_base + GPT_O_CTL) |= 0xC;
    HWREG(timer_base + GPT_O_CTL) |= 0x0; // positive edge
    HWREG(timer_base + GPT_O_TAMR) |= 0x200;
// 7. Load the timer start value into the GPTM Timer n Interval Load Register (GPT:TnILR…
    ).
    TimerLoadSet(timer_base, TIMER_A, TIMER_LOAD_VAL);
// 8. Load the GPTM Timer n Match Register (GPT:TnMATCHR) with the match value.
    TimerMatchSet(timer_base, TIMER_A, TIMER_LOAD_VAL/2); // to get 50% duty cycle
// 9. Set the GPTM Control Register (GPT:CTL) TnEN bit to enable the timer and begin …
    generation of the output PWM signal.
    TimerEnable(timer_base, TIMER_A);

}
```

C

Sensor data retrieval :Listing

The code snippet below shows how the data bits from the sensor are combined to form a 20-bit acceleration value according to the figure 3.4. The value that is stored in the *xdata, ydata and zdata* variables are then multiplied by the scaling factor corresponding to its range setting. The function *Sen_Single_Byte_Read* takes the source register address and a pointer to the destination address as its parameters. This function accesses the SPI via the SPI driver in order to move data to the microcontroller and then to the memory for storage.

```c
Sen_Single_Byte_Read(XDATA3,(int8_t*)&xdata3);
Sen_Single_Byte_Read(XDATA2, (int8_t*)&xdata2);
Sen_Single_Byte_Read(XDATA1, (int8_t*)&xdata1);
xdata =(int)xdata3<<12|(int)xdata2<<4|(int)xdata1>>4;
if(xdata & (1 << 20 - 1))
    xdata = xdata - (1 << 20);

Sen_Single_Byte_Read(YDATA3, (int8_t*)&ydata3);
Sen_Single_Byte_Read(YDATA2, (int8_t*)&ydata2);
Sen_Single_Byte_Read(YDATA1, (int8_t*)&ydata1);
ydata = (int)ydata3<<12|(int)ydata2<<4|(int)ydata1>>4;
if(ydata & (1 << 20 - 1))
    ydata = ydata - (1 << 20);

Sen_Single_Byte_Read(ZDATA3, (int8_t*)&zdata3);
Sen_Single_Byte_Read(ZDATA2, (int8_t*)&zdata2);
Sen_Single_Byte_Read(ZDATA1, (int8_t*)&zdata1);
zdata = (int)zdata3<<12|(int)zdata2<<4|(int)zdata1>>4;
if(zdata & (1 << 20 - 1))
    zdata = zdata - (1 << 20);
```

D
Clock frequency stability distribution

From the discussion on sensor synchronization, separate clock sources were required to act as master clocks for the accelerometer sensor. In order to validate the approach, two clock sources were considered, namely: General purpose timer clock and function generator. The validation of these clocks were presented in section 3.5 with the help of universal frequency counters and eye patterns. Every oscillator has an offset error and a random error. An offset error is more or less the same for given period of measurement. But the random error is the frequent fluctuation of frequency. A more detailed information can be obtained by the distribution of the frequency variation. The plots below show the distribution of random error around the offset error for the nominal frequency of 1.024MHz.

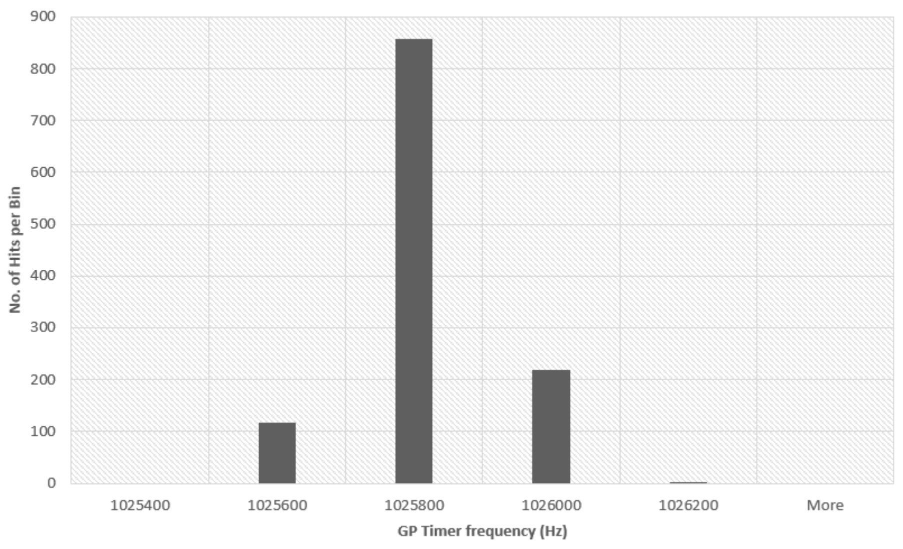


Figure D.1: Frequency Variation of General Purpose Timer clock

The figure D.5 shows the three function generators used for emulating the clocks and the external synchronization signal for the sensors. The clocks are set at a nominal frequency of 1.024MHz, 3 V peak-to-peak and at a duty cycle of 50%. The external synchronization pulse is given by a 1Hz square wave signal.

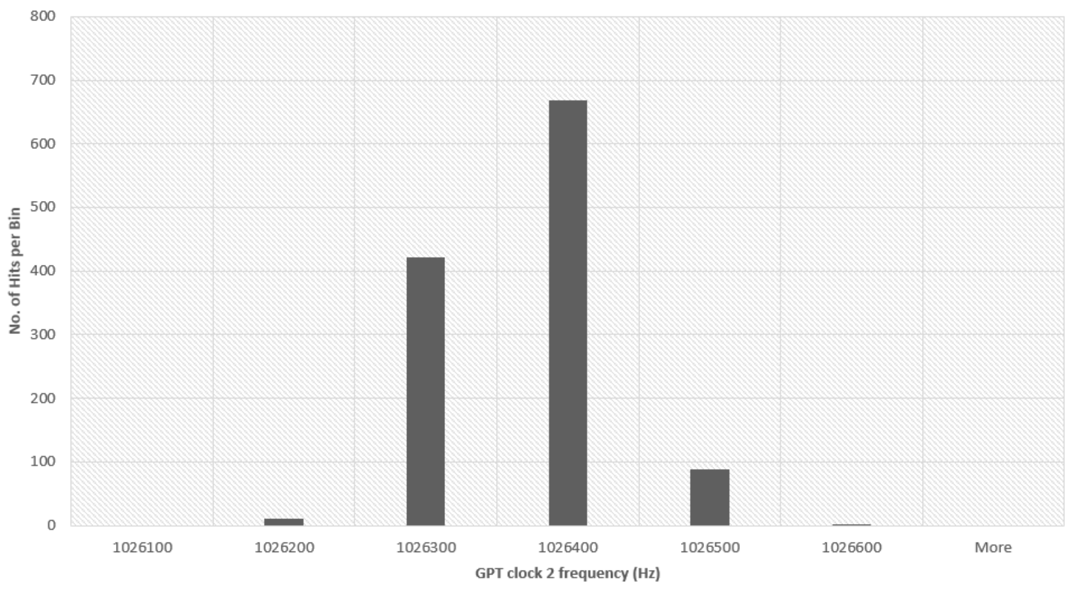


Figure D.2: Frequency Variation of General Purpose Timer clock

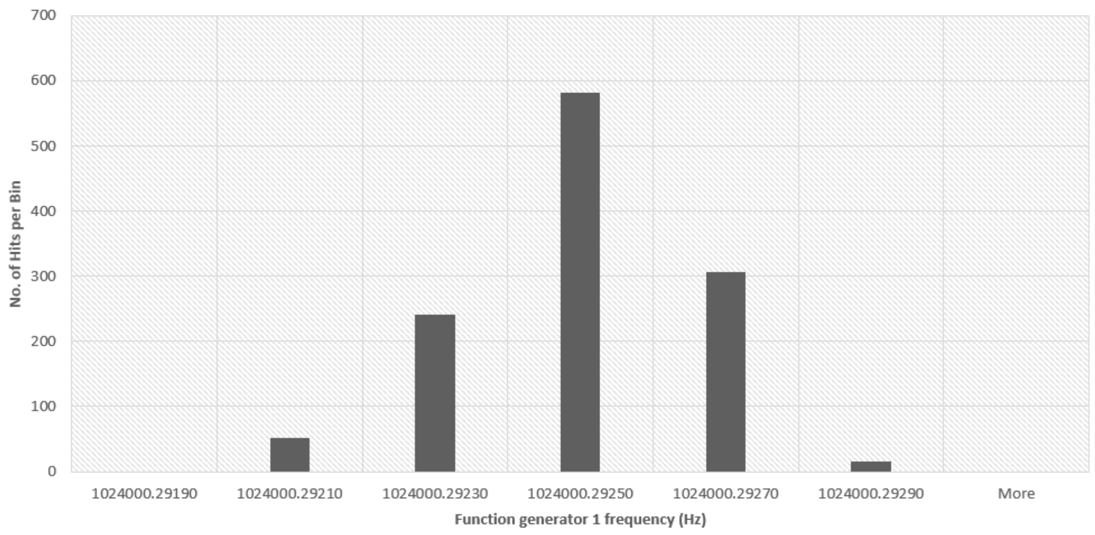


Figure D.3: Frequency Variation of Function generator 1

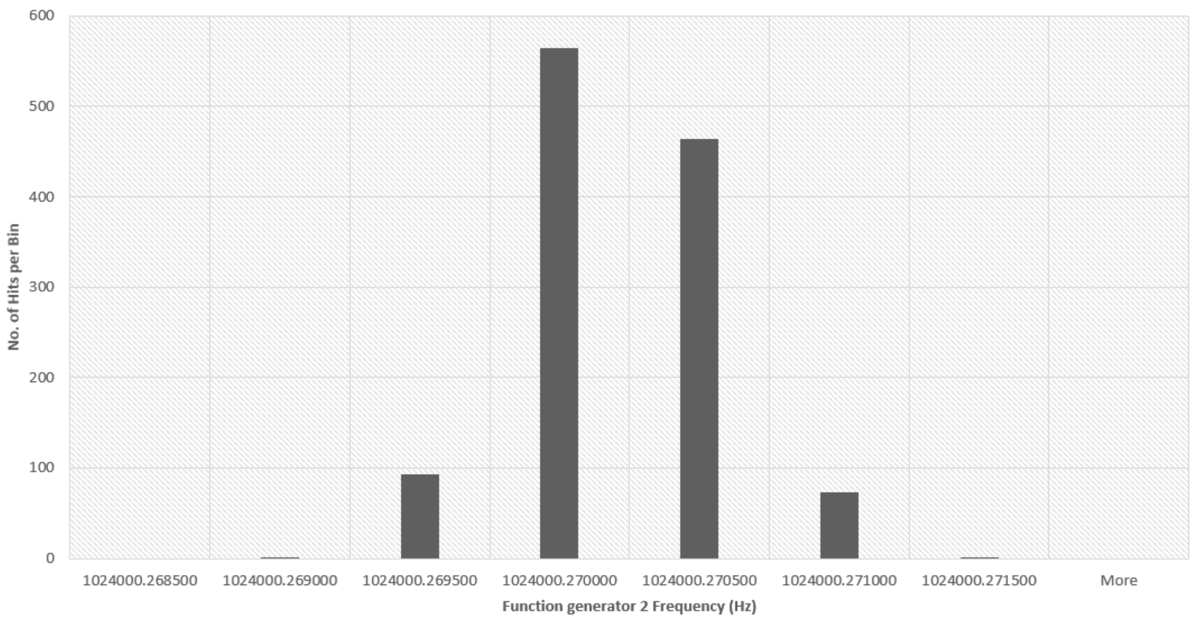


Figure D.4: Frequency Variation of Function generator 2

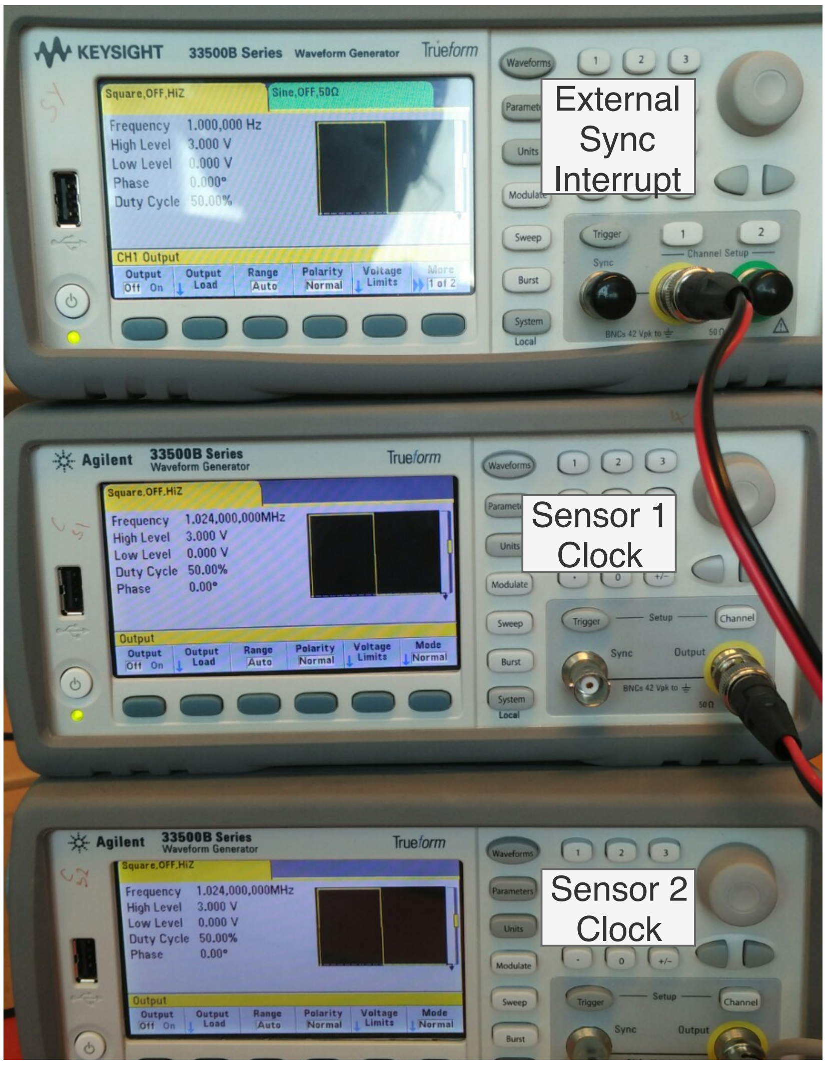


Figure D.5: The Function generators used as clock and external synchronization interrupt sources

E

Subsequent synchronization :Listing

The following MATLAB code listing is used to perform subsequent synchronization on the obtained acceleration data from two sensors for the same input signal as discussed in section 4.3.2.

```
clear;
clc;
close all;

% Load data

filename = "sen1shaker2.csv";
raw_data1 = csvread(filename);
t1=raw_data1((9000:11000),1)/48;
x1=raw_data1(:,2)/256000;
y1=raw_data1(:,3)/256000;
z1=raw_data1((9000:11000),4)/256000;
filename = "sen2shaker2.csv";
raw_data2 = csvread(filename);
t2=raw_data2((9000:11000),1)/48;
x2=raw_data2(:,2)/256000;
y2=raw_data2(:,3)/256000;
z2=raw_data2((9000:11000),4)/256000;

z1=z1-mean(z1);
z1=z1/max(z1);
z2=z2-mean(z2);
z2=z2/max(z2);

ts1=timeseries(z1,t1);
ts2=timeseries(z2,t2);

[ts1,ts2]=synchronize(ts1,ts2,'Uniform','InterpMethod','linear');
figure;
plot(ts1.Time,ts1.Data);
hold on
plot(ts2.Time,ts2.Data);

% Cross-correlation

T1=z1;
T2=z2;
Fs1=1000;
Fs2=1000;
Fs = 1000;
```

```

[P1,Q1] = rat(Fs/Fs1);          % Rational fraction approximation
[P2,Q2] = rat(Fs/Fs2);          % Rational fraction approximation
T1_res = resample(T1,P1,Q1);        % Change sampling rate by rational factor
T2_res = resample(T2,P2,Q2);        % Change sampling rate by rational factor

dt = 1/Fs;
t = 0:dt:dt*(length(T1_res)-1);



figure
hold on;
plot(t1,T1,'LineWidth',2)
plot(t2,T2,'LineWidth',2)
title('Original Data');
legend('Sensor 1 z-axis','Sensor 2 z-axis');
xlabel('Timestamp(microseconds)');
ylabel('Acceleration (g)');


figure
hold on;
plot(t,T1_res,'LineWidth',2)
plot(t,T2_res,'LineWidth',2)
title('Resampled Data');
legend('Sensor 1 z-axis','Sensor 2 z-axis');
xlabel('Times(microseconds)');
ylabel('Acceleration (g)');

[C1,lag1] = xcorr(T1_res,T2_res);


figure
hold on;
plot(C1)
title('Cross-correlation')
[¬,I] = max(abs(C1));
SampleDiff = lag1(I);

timeDiff = SampleDiff/Fs


figure
hold on;
plot(t,T1_res,'LineWidth',2)
plot(t+timeDiff,T2_res,'LineWidth',2)
title('Aligned signal');
legend('Sensor 1 z-axis','Sensor 2 z-axis');
xlabel('Time(microseconds)');
ylabel('Acceleration (g)');
```