The Pennsylvania State University

The Graduate School

Capital College

A ROLL DOWN THE LANE

MEASURING BOWLING BALL DYNAMICS FROM THE INSIDE
INTRODUCING THE *REVMETRIX* SYSTEM

A Master's Project and Paper in Engineering Science

by

Donald J. Hake II

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering

October 2014

The Master's paper of Donald J. Hake II was reviewed and approved* by the following:

Sedig Agili
Professor of Electrical Engineering
Program Coordinator, Master of Engineering in Engineering Science

Eugene Boman
Associate Professor of Mathematics

Aldo Morales
Professor of Electrical Engineering

Linda Null
Associate Professor of Computer Science

Seth Wolpert
Associate Professor of Electrical Engineering
Paper Advisor

*Signatures are on file in the Graduate School.

ABSTRACT

This paper summarizes the design, development, and testing of a performance analysis system intended to directly capture the various dynamic forces that a bowling ball experiences, from within the bowling ball itself. The sensor module at the heart of the system is designed to reside in the ball, at the bottom of a finger hole, underneath a finger insert. The sensing capabilities of this *in situ* module enable it to not only capture the reaction of the ball as it rolls down the lane, but also to record the motion of the ball as the bowler goes through their[1] approach, the impetus the ball experiences as the bowler applies "lift" and "turn" to it as part of their release motion, the ball's reaction to being lofted as the result of release, as well as the reaction of the ball as it drives through the pins and into the pit.

The sensor module presented in this paper is a greatly updated and enhanced version of the *SMARTDOT* module, originally presented in the author's Computer Science Master's thesis, "A Performance Analysis System for the Sport of Bowling" (Penn State Harrisburg, May 2002). The system presented in that thesis captured the changing ambient light level at the sensor module as the ball rotated under the overhead lighting, and then later inferred the angular velocity of the ball from that waveform. That sensing methodology was utilized (at the time the research was conducted) due to the relatively inexpensive nature of light sensing technology compared to the cost of using an onboard accelerometer to directly measure the ball's motion.

In the interim, however, micro-machined accelerometers have undergone a dramatic drop in size, cost, and power requirements, while their functionality and overall performance have increased considerably. In addition, highly configurable "system-on-a-chip" microprocessors have become readily available that offer tremendous improvements to the original *SMARTDOT* module's capabilities, while also reducing the overall production cost of the module.

As such, this paper presents a next-generation sensor module design based upon a state-of-the-art system-on-a-chip microprocessor that interfaces with a 3-axis accelerometer, ambient light sensors, and expanded non-volatile memory for data storage and retrieval. The sensor module has been designed from the outset to be an inexpensive, consumer-based product that is user installable and replaceable. Careful consideration has also been given to creating a module whose presence and operation is completely transparent and unobtrusive to the user, with additional consideration applied to the module's physical design so that its presence in the ball has minimal impact on the ball's static and dynamic balance.

This presentation includes the design and implementation of a fully functional *in situ* sensor module: the physical design constraints; the hardware and sensor requirements, schematic, and PCB layout; the microprocessor configuration; the embedded software requirements, design, and implementation; as well as a summary of the module's performance under real-world conditions.

Breakdown and analysis of the raw data waveforms is included, along with a presentation of the preliminary algorithm development for reliably extracting and deriving a set of useful bowling metrics from the collected sensor data. The resulting raw data filtering and waveform analysis algorithms rely heavily on the application of Fast Fourier Transforms in combination with techniques taken from Wavelet Theory.

[1] In the absence of a standard set of gender-neutral pronouns, the author uses "they", "them", and "their" as the gender-neutral forms of "s/he", "him/her", and "his/her" throughout the document.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

It has been 16 years since I started graduate work at Penn State Harrisburg, and a dozen years since I graduated with my Master's in Computer Science. Little could I have anticipated that taking COMP 432 (Advanced C++) from Dr. Bui over the Summer of 1998 would lead me on this extensive journey at Penn State Harrisburg – 80 credits, two Master's degrees, the opportunity to teach a couple of courses, and the chance to meet such wonderful faculty, in both the Computer Science and the Electrical Engineering departments. I owe a debt of gratitude that can't be repaid, for this experience has truly been transformative. Through these last 2^4 years (seems appropriate that it should be an integral power of 2), I have been able to explore what it is that truly makes me joyful – the opportunity and the knowledge to pursue my own curiosities, combined with the chance to interact with both faculty and students in such a fulfilling way.

My gratitude goes out to the following faculty members at Penn State Harrisburg, who have been nothing but enlightening, challenging, instructive, and inspirational to me throughout my extended academic adventure. Thank you for sharing your knowledge, your teaching skills, for opening up a world of possibilities for me, and for your constant encouragement and support.

Dr. Aldo Morales: For showing me the wonders of digital signal processing, FFTs, and especially Wavelet theory, which still seems so mathemagical to me, at times.

Dr. Eugene (Bud) Boman: For teaching me the matrix theory that underpins so much of the FFT, Wavelet theory, and all of their calculations.

Dr. Seth Wolpert: For not only being willing to be my thesis advisor and suffer through reviewing a second iteration of the physics of bowling, but also for showing me the wonders of neural networks, and the intricacies behind VLSI design and solid state manufacturing.

Dr. Linda Null: For also agreeing to put herself through reading and reviewing a second 100+ page treatise on this topic, as well as being a good friend and mentor.

Dr. Thang Bui: For sharing his extraordinary insight into evolutionary computation and genetic algorithms. And for the line, "This is Computer Science – we talk about it, we don't do it."

Dr. Sukmoon Chang: For extending my knowledge of artificial intelligence.

Dr. Jeremy Blum: For introducing me to the methods for mobile application development, as well as providing me with a new model of decorum for deftly handling those inevitable improvisational moments that arise in the classroom with both humor and grace.

My thanks also go out to Becton Dickinson Diagnostic Systems for fully funding this degree, and to all of the managers that have supported me along the way: Scott Shindledecker, Cheryl Abel, Rick Altekruse, and Vince Federico.

And my heartfelt gratitude goes out to my lovely, loving, understanding, and patient wife Sandy, who for the second (and likely not the last) time in our marriage has suffered through my extended periods of isolation in my office, bowling balls bouncing off furniture in the basement, and my borderline(?) OCD fascination with building the *SenseModule*, and my boundless gEEky excitement at my first site of the 3-axis acceleration waveforms.

Section I: Introduction and Background

1.1 Statement of the Problem[2]

The research presented in this paper is a continuation, refinement, and extension of the author's previous development of the original *SMARTDOT* system presented in his Master's Thesis, "A Performance Analysis System for the Sport of Bowling" [1].

Bowling is often considered a game of accuracy, but it is actually a game of errors. The goal of any experienced bowler is to find the optimal combination of style, equipment, and lane adjustments that affords the greatest margin of error while still allowing the bowler to consistently deliver the ball to the pocket with sufficient force, angle, and "action" to generate strikes. Success with this strategy requires a combination of factors: the bowler's natural talent and ability, refined by a generous amount of coaching and practice; experience with "reading" lane conditions and making adjustments to the inevitable changes in those conditions; and the selection and use of the proper equipment, e.g., picking the right bowling ball for the conditions.

Bowling balls are available in a variety of weights, balances, hardnesses, and surfaces. Those four variables, combined with the bowler's style, determine when and how much the ball hooks, and how hard it hits the pins. The bowler selects a bowling ball from their collection based on his or her bowling style and the current lane conditions, which are determined by the lane oil distribution.

As a bowling ball rolls down a lane, a great deal of friction is generated between the ball and the lane surface. The linear velocity of the ball can approach 20 mph, while the angular velocity can easily exceed 300 rpms. To limit the wear on the lane, as well as to make the lane "playable", special lane-dressing oil is regularly applied to the lane. Lane oil is generally applied with a varying density both across and down the lane which affects the "playability" of the lane, making it easier or more challenging for the bowler to consistently deliver the ball to the strike pocket.

As bowling balls repeatedly roll through the lane oil, they redistribute that oil over time, changing the oil pattern as a bowling session progresses. The effects of that change can be quite noticeable, sudden, and dramatic. A primary concern for all bowlers is to quickly identify those changes, and correctly adjust to that changing oil pattern.

The bowling ball is the bowler's "oil sensor" for determining where the lane oil is (and isn't), as well as how that oil distribution is changing. Based solely on observing the ball's reaction to the lane, the bowler adjusts to the ever-changing lane condition by drawing upon their past experience with the results of various adjustments made under similar circumstances. Those adjustments usually involve lateral changes in the starting location on the approach and/or the target on the lane, an increase or decrease in the speed of the ball, and/or a switch to a ball that hooks more, or less, or sooner, or later, etc. The bowler may also opt to change the amount of

[2] Portions have been excerpted and/or paraphrased from the Abstract and Problem Statement from [1], with appropriate edits and updates applied.

turn, lift, and/or loft they apply to the ball during release, with the intention of changing the amount the ball hooks.

If the bowler does not release the ball with consistent amounts of speed, turn, lift, and loft (the variables that directly affect the ball's reaction with the lane), it is particularly difficult to accurately assess the condition of the lanes, let alone how that condition is changing. It has always been difficult to accurately quantify a bowler's relative level of consistency. It has been equally difficult to quantify and compare the relative performance of different types of bowling balls.

Current technology offers precious little to the serious bowler in search of ways to analyze and improve their game. All of the existing methods currently available to the bowler rely on acquiring some type of "external" view of the ball (from the bowler's perspective) as it rolls down the lane. The goal of such systems is to quantify the various factors (release velocity, rotation rate, ball loft distance, etc.) that contribute to the ball's reaction and, ultimately, to the bowler's performance. However, not only are those externally-based methods time-consuming, inconvenient, and/or expensive to install and use, but they each have their own inherent limitations resulting from their external view of the ball.

As of this writing, no devices of the nature described in the current literature have been brought to market with regards to bowling [1], [4], [7]. Between the inconvenience, expense, and narrow availability of external (instrumented lane) solutions, and the dearth of internal solutions, it remains particularly difficult for a bowler to accurately and adequately assess the various impacts that changes in bowling style and bowling equipment have on their game. Without such timely and consistent feedback on changes in wrist and hand position, arm swing, stance, and grip, as well as changes in equipment (ball type, weight, balance, and/or surface), the bowler generally participates in a guessing game when assessing the effectiveness and usefulness of any of these considerations.

At the time of the development of the first *SMARTDOT* sensor module, solid-state accelerometers were prohibitively expensive for use in a low-priced (sub-$50.00 MSRP) consumer electronics device. As such, the original *SMARTDOT* module relied on a far less expensive solution based on an ambient light sensor, and a piezoelectric film sensor that doubled as the start-up circuit and the impact sensor. The ambient light sensor was used to detect the varying light level at the sensor module as the ball rolled down the lane, while the module's microprocessor sampled the light waveform and stored the resulting sensor data in external EEPROM for later analysis.

Subsequent analysis of the captured ultimately waveform revealed that it was difficult to reliably deduce the instantaneous angular velocity of the ball from the sampled light data. There was also no practical method to verify that the extraction techniques that the author proposed and eventually implemented had accurately reconstructed the motion of the ball.

Figure 41 in Appendix A (page 116) depicts the layout of a typical bowling lane, including the overhead lighting sources, and the generalized ambient light waveform that both the original *SMARTDOT* module and the next-generation module collect.

With the passage of time, sufficiently inexpensive, 3-axis, micro-machined accelerometers have become readily available within a price range that makes them practical for inclusion in the sensor module application. Also, mixed-signal semiconductor technology has advanced to the point where small, low-power system-on-a-chip microprocessors are now available that greatly reduce the component count (and cost), while adding a great deal of functionality and processing power to the application.

This paper summarizes the design and development of a greatly updated and enhanced version of the original *SMARTDOT* module, utilizing an 8-bit system-on-a-chip microprocessor, a 3-axis micro-machined accelerometer, a light level sensor, and an ambient light-based start-up circuit. The enhanced capabilities of this next-generation sensor module allow it to capture an "internal" view (from the bowling ball's perspective) of the dynamics that the ball experiences throughout its journey to the pins. The new sensor module not only captures the interaction of the ball with the lane surface as it rolls down the lane, but also captures the motion of the ball as the bowler goes through their approach, the impetus they apply to the ball at the time of release, and the ball's reaction as it drives through the pins.

The new sensor module is part of the overall *REVMETRIX* system, consisting of three components:

- The *SenseModule* (*SM*): An *in situ* sensor and data collection module that resides in the ball at the bottom of a finger hole, underneath a finger insert, as shown in Figure 1.
- The *ComModule* (*CM*): A wireless communications module that serves as the interface between the *SM* and the *RevMetrix* application. Currently, the *ComModule* connects to the host platform (a PC) via a USB cable, but could be connected wirelessly in the future.
- The *RevMetrix* application (*RMApp*): A multi-platform (PC, tablet, smart phone) software application that uploads, archives, analyzes, and displays the data captured by the *SenseModule* and retrieved through the *ComModule*.

In the interest of brevity, the terms *SenseModule* and *SM*, *ComModule* and *CM*, and *RevMetrixApp* and *RMApp* are used interchangeably throughout the paper.

1.2 *SenseModule* Installation

Figure 1 shows a cut-away view of the *RevMetrix SenseModule*, as installed in the bowling ball underneath a finger insert in an existing finger hole. The case that will hold the *SenseModule* and battery has not yet been designed, and is not shown.

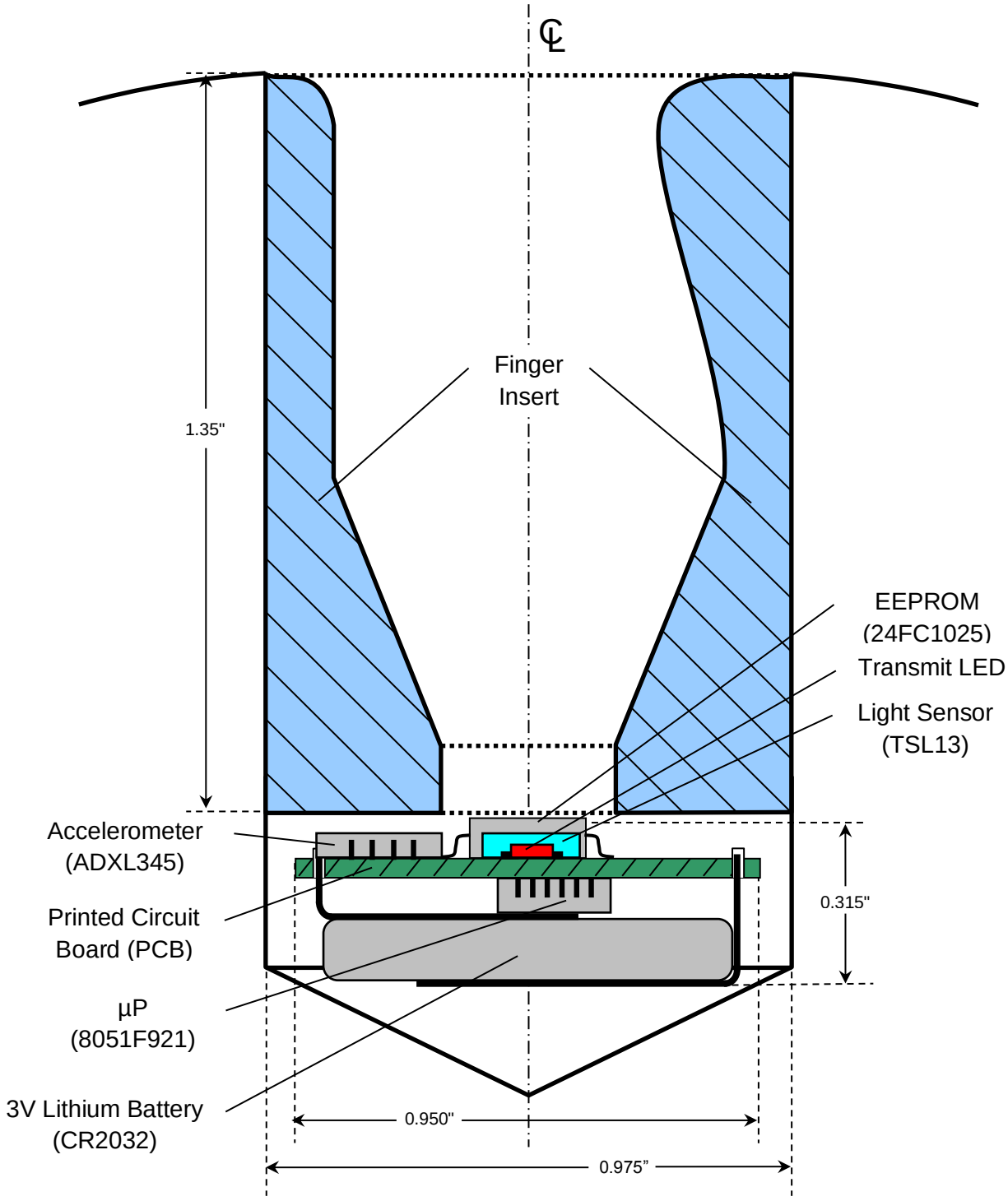

Figure 1: *SenseModule* Cut-Away View

1.3 A Summary of the Physics of Bowling[3]

Bowlers attempt to make a bowling ball hook in order to generate more pin action. Industry research has shown a direct correlation between a bowling ball's angle of entry into the strike pocket, and the chance of generating a strike [1]. During release, the bowler imparts an axis of rotation to the ball that is turned and/or tilted away from normal, which is intended to make the ball hook towards the pocket. It is the interaction of the linear and angular velocities of the ball with the frictional force acting between the ball and the lane that causes the ball to hook.

In order to control the amount the ball hooks, the bowler releases the ball with various combinations of speed, loft, lift, turn, and tilt. Those five terms are defined below:

1) **Speed:** The initial linear velocity of the ball as it leaves the bowler's hand. The higher the initial velocity, the less opportunity for the ball to hook, but the more energy the ball can impart to the pins. The bowler varies the initial velocity of the ball by adjusting the push-away height of the ball at the start of their approach, their approach speed, and/or their arm swing speed.

2) **Loft:** The longitudinal distance the ball travels before making initial contact with the lane. The longer the loft distance, the less opportunity the ball has to hook. The bowler can control the loft distance through the release velocity and release point of the ball: an earlier release directs the ball parallel to the lane surface; a later release directs the ball upwards, away from the lane surface. For any given loft height, a higher release velocity causes the ball to travel further before striking the lane.

3) **Lift:** Lift is the initial angular velocity that the bowler applies to the ball during release. It causes the ball to begin rotating before hitting the lane. The more lift the bowler applies during release, the higher the initial angular velocity of the ball. With the proper axis turn and tilt, more lift increases the potential amount the ball will hook.

4) **Turn:** The amount that the ball's axis of rotation is rotated away from normal (orthogonal to the direction of travel). The ball will eventually hook toward the direction that the axis of rotation is turned. The more the bowler turns the axis away from normal (with respect to the direction of travel), the more potential the ball has to hook. If the initial axis of rotation is normal to the initial direction of travel, and parallel to the lane surface, the ball will not hook, except due to weight imbalances in the ball.

5) **Tilt:** The amount that the ball's axis of rotation is rotated with the respect to the lane surface. More tilt implies more "spin" – the extreme would be that the ball spins like a top (the rotational axis is perpendicular to the lane surface). Increased axis tilt delays the ball from hooking in the direction of the axis turn.

After the bowler releases the ball, the only external force of any significance that acts upon the ball is the force due to friction generated between the ball and the lane (this force varies a great

[3] Excerpted and/or paraphrased from Section 1.2 "Introduction to the Physics of Bowling" in [1], with appropriate edits applied.

deal due to the variations in oil distribution on the lane). A bowler invariably releases a ball with an initial linear velocity (speed) that is greater than that which would result from the ball's initial angular velocity (lift). In other words, the distance the ball travels during one complete revolution is greater than the circumference of the ball (the ball is skidding or sliding as it is also rotating).

As long as the ball is skidding, friction acts to transfer some of the ball's linear momentum into angular momentum, decreasing the linear velocity while increasing the angular velocity. Thus, as the ball "revs up", it slows down. If the linear and angular velocities completely resolve themselves, the ball is no longer skidding (it has *rolled out*), and the frictional force now causes the angular and linear velocities of the ball to decrease in direct proportion to each other [5].

There is also a significant internal dynamic force that acts upon the ball; the changing rotational inertia of the ball due to imbalances in specially shaped weight blocks designed for just such a purpose. Such weight blocks can delay or accelerate the reaction of the ball with the lane [6].

A small percentage of the ball's linear momentum is also lost due to wind resistance, and the heat, noise, and vibration generated as the surfaces of the ball and the lane rub against each other.

The 3-axis accelerometer and the ambient light sensor are used to capture the following forces acting upon the ball:

1) The impetus the bowler applies to the ball, from the start of the approach to the moment of release: the speed of the approach, the speed of the ball through the arm swing, the lift, turn, and tilt the bowler applies to the ball during release.
2) The moment of release, the moment(s) of impact with the lanes, and the various impacts with the pins.
3) The ball's 3-axis angular velocity due to the frictional force between the ball and the lane surface from its initial impact with the lane, to the time it falls into the pit at the end of the lane.

By measuring the time between release and impact, and noting the duration of each revolution of the bowling ball, it is possible to derive the angular velocity of the ball for each revolution, and the energy necessary to induce this change in angular velocity. The average linear velocity of the ball and the changes in the ball's angular velocity during the course of the shot can be combined with certain assumptions regarding momentum and energy conservation and friction to derive the instantaneous linear velocity and longitudinal location of the ball. Using the same criteria, it is also possible to derive the varying frictional force between the ball and the lane.

The mathematical details of those calculations are presented in the authors' original paper [1], and have been included in the appendices of this paper.

1.4 Scope of the Paper/Project

The development of the *SenseModule*, the performance results, and the analysis of the collected data is presented in the following sections:

- Section II presents the *SenseModule* design constraints, hardware requirements, and module implementation.
- Section III presents the *SenseModule* embedded software requirements, design, and implementation.
- Section IV presents the *SenseModule* performance results, including examples of the collected sensor data.
- Section V presents the initial data analysis algorithm development, error analysis, along with suggestions for future work.
- Section VI presents a summary of the entire project.

Section II: *SenseModule* Hardware Requirements and Implementation

The *REVMETRIX* system consists of a microprocessor-based data collection sensor module referred to throughout this paper as the *SenseModule* (*SM*), which interacts with an external wireless communications module referred to as the *ComModule* (*CM*). The *SM* transfers the raw sensor data that it collects to the *CM*, which is connected to a smartphone, tablet, or PC for data archival, analysis, and presentation by the *RevMetrix* application (*RMApp*).

The *SenseModule* is designed to collect accelerometer and ambient light data with a sufficient granularity so that the *RevMetrix* application can provide accurate and meaningful analysis of that data. As with the first version of the *SMARTDOT* sensor module, the key to the feasibility of the system has been the initial development of the *SenseModule*. Thus, the initial design, development, and implementation of the *SenseModule* have been directed towards collecting that data so that an initial analysis of the data could be performed. This section discusses the design assumptions, constraints, and capabilities of the *SenseModule*.

As far as the author has been able to discover, the manner in which the *SenseModule* captures the data (autonomously, with an inexpensive module, from within an *unaltered* bowling ball) has never before been accomplished, although it must be noted that this not the only time that this data has been collected. The closest reference found in the literature was published within two months after the conclusion of the testing performed for this paper. The "IMU" (Inertial Measurement Unit) specified in [7] utilizes a 3-axis accelerometer and two rather expensive angular-rate gyros, along with a wireless transmission unit. It is placed in a separate 1.25" hole, drilled to a depth of ~2.5", and such an installation could certainly impact the balance characteristics of the ball. Additional references appear in the literature for "the first instrumented bowling ball" [1] and to "performance analysis with an instrumented bowling ball" [4] , but those instances rely on measurements obtained from bowling balls so highly altered and weighed-down with force transducers as to be rendered unusable for normal play.

2.1 Physical and Functional Requirements

2.1.1 Physical Design Constraints

The *SenseModule* must conform to a varied collection of physical, economic, and electronic constraints. These design constraints are similar to that of the original *SMARTDOT* module.

1) **Transparent:** The presence and operation of the *SenseModule* must not be apparent to the bowler. The *SM* must start up, operate, and shut down automatically, without any user intervention – the bowler should not be able to detect the presence of the *SM*, and the operation of the *SM* should not impact the bowler's normal routine in any way.

2) **Small and light weight:** The *SenseModule* must be sufficiently small enough that it can be located at the bottom of a finger or thumb hole. It must also be as lightweight as possible to minimize its impact on the static and/or dynamic balance of the ball.

Nominally, it should have a similar weight as any excess material that must be removed from the finger hole in order to install the module.

3) **Low cost:** The *SenseModule* must be inexpensive relative to the cost of a state-of-the-art bowling ball, since the bowler will likely need multiple *SM*'s, as it will be inconvenient to switch a single *SM* between multiple bowling balls.

4) **Low power:** Since the *SenseModule* is battery powered, and the battery comprises a significant fraction of the SM's size and weight, the *SM* should be designed to draw minimal current at all times. The *SM* must spend the vast majority of its time in a micro-power standby mode, and have a means for automatically detecting the bowler's presence, release of the ball, and shutdown of sampling to conserve battery life.

5) **Replaceable:** Ideally, the *SenseModule* should be user-installable, and the battery should be rechargeable and/or user-replaceable.

2.1.2 Sensors

The *SenseModule* must be able to sense the start-up condition, the motion of the ball while in the bowler's hand, the bowler's release of the ball, the ball's impact with the lane and the pins, and the rotation of the ball. The *SM* must be able to accurately sense the passage of time, with microsecond resolution, and record time-stamped digitized waveforms of the various sensors. It must also automatically detect the presence of, and communicate with, the *ComModule*.

The original *SMARTDOT* module has two sensors: a piezoelectric film sensor used to detect start-up/release and the ball's impact with the lane and the pins; and an ambient light sensor used to detect the moment the bowler released the ball, and then to infer the rotation of the ball through the changes in the ambient light level, which only occur after release. The light sensor also doubles as the infrared receiver for communication with the *COMM* wand.

The new *SenseModule* is based upon a 3-axis accelerometer, although the *SM* still has a light sensor. With the inclusion of an accelerometer, the *SM* is able to directly sense the motion and rotation of the ball and, by extension, certain movements of the ball during the bowler's approach and release. The accelerometer is also able to sense the ball's impact with the lane and the pins. The *SM* periodically samples the three axes of the accelerometer, and stores those readings in non-volatile memory for later transfer to the *ComModule* via an infrared interface.

The ambient light sensor has been carried over from the original *SMARTDOT* module. It is used to detect the presence of the bowler's finger in the finger hole (and the presence of the *CM*), and serves as the optical receiver for communications purposes. The *SM* also periodically samples the light sensor and stores those readings along with the accelerometer data. As part of follow-up research to the original paper, the ambient light data will be correlated with the accelerometer data to ascertain the relative accuracy of the methods implemented in the previous paper for inferring the motion of the ball from the captured ambient light waveform.

The inclusion of an accelerometer mostly supplants the function of the piezoelectric film sensor used in the original *SMARTDOT* module. However, the piezo film also served as a passive start-up

sensor for the *SMARTDOT* module. Unfortunately, the accelerometer draws too much current to be used in the start-up circuit. Therefore, a different method for sensing start-up has been developed that eliminates the need for the piezo film sensor.

The new light-based start-up circuit indirectly detects that the bowler has placed their finger in the finger hole (or placed the *CM* over the hole). Since the existing ambient light sensor also draws too much current to remain constantly powered, the start-up circuit uses a separate, low-cost photo-transistor as part of a micro-power circuit that continuously monitors the light level reaching the *SM*. When the light level falls below a certain threshold for a given amount of time, the start-up circuit, combined with a micro-power comparator on the microprocessor, wakes up the rest of the *SM*. This new light-based start-up circuit draws less than 2 μA of current, and rejects the vast majority of spurious light pulses. It may eventually be possible to combine the two light sensing circuits, but that was not a goal for this version of the module.

2.1.3 Microprocessor

Since the *SenseModule* must be small, low cost, and draw little current, a small form-factor, 8-bit microprocessor made the most sense for this application. Since the development of the original *SMARTDOT* module, small, low-cost, versatile, and powerful system-on-a-chip 8-bit microprocessors have become readily available, and sufficiently inexpensive for this application.

Such microprocessors now contain comparators, analog-to-digital converters, built-in programmable clock sources, on-chip non-volatile, in-system writeable flash memory, expanded code memory and RAM, and many other functions.

Ideally, the microprocessor for the *SM* must possess the following qualities:

1) Small form factor
2) Low cost
3) Low power (whether in standby mode, idle mode, or while executing code)
4) Internal reset circuitry
5) Internally generated system clock (no external components required)
6) A micro-power real time clock function
7) On-board analog functions such as low-power comparators, a multi-channel ADC, a voltage reference, programmable constant-current source, etc.
8) Configurable port pins
9) In-system circuit emulation and flash programming
10) Hardware support for serial EEPROM and serial communications
11) Sufficient on-board RAM for data capture buffers, serial communication buffers, etc.

2.1.4 External Memory

The *SenseModule* collects sample data from four sensor channels (the ambient light sensor, and the three accelerometer axes), and must sample at a rate sufficient to accurately detect transient occurrences such as the ball's impact with the lane and impact with the pins. The sample memory should also be large enough to accommodate all 12 possible first shots of each frame (the first 9 frames, plus 3 possible shots in the 10^{th} frame), and ideally at least one game's worth of data, a maximum of 21 shots (2 each in the first 9 frames, plus 3 in the 10^{th} frame).

The memory must be non-volatile, and draw little current while not being accessed. It must also easily interface to the microprocessor to limit the execution time (and battery power) spent on accessing the memory.

2.1.5 Communications

The *SenseModule* captures the raw sensor data and stores it in external non-volatile memory until such time as it can transfer that data to the *ComModule*. The transfer medium must be wireless (non-contact) since the *SM* resides at the bottom of a finger hole. Given that the *SM* already has an optical receiver, and requires just a single LED for transmission, the most cost-effective method for the wireless transfer is via an optical interface (visible or infrared). The communications requirement imposes additional response time challenges on the ambient light sensor. Whereas the ambient light waveform may vary at a 10-20 Hz rate while the ball is rolling, the ambient light sensor must also be fast enough to accept firmware updates of 4-8 KB in a timely fashion (no more than a few seconds).

2.2 *SenseModule* Component Selection

Major component selection is presented below, along with a summary of each component. Links to the manufacturer's data sheets are provided in the bibliography. A theory of operation for the overall *SenseModule* circuit, as well as for each of the individual sub-circuits of the *SM* follows the schematic. The current schematic diagram for the *SM* is given in Figure 2 on the following page. The layout of the printed circuit board (PCB) used to build the *SenseModule* prototypes is shown in Figure 3 (top) and Figure 4 (bottom).

- **Start-Up Phototransistor (T1):** Optek Technology OP521 Phototransistor [27]
  - Spectral Responsivity: 550-1060nm (25%), 910nm (peak)
  - Operating Range: -25 to 85°C
- **Microprocessor (IC1):** Silicon Laboratories 8051F921 8-Bit Flash Microcontroller [23]
  - Programmable internal oscillator, 3 to 24.5 MHz, ± 2% accuracy
  - Real-Time Clock (*smaRTClock* w/32kHz crystal, 0.5 µA supply current)
  - On-chip debug circuitry for full-speed in-system debugging
  - 10-Bit 300 Ksps ADC, 15 single-ended or differential inputs
  - Two, low-power (< 0.5 µA) comparators, available even when µP is stopped
    - Programmable hysteresis and response time
    - Configurable as interrupt or reset sources
  - 4352 bytes on-board data RAM (256 + 4KB XRAM)
  - 32 KB flash program memory, in-system programmable in 1024-byte sectors
  - Four general purpose 16-bit counter/timers
  - UART, SMBus (I^2C), and SPI serial ports
  - Supply Voltage: 0.9 – 3.6 volts (on-board DC-DC converter for 0.9 to 1.8 volts)
  - Supply Current:
    - 4.1 mA at 25 MHz
    - 11 µA at 32 kHz
    - 0.6 µA in stop mode, w/*smaRTClock* enabled
    - 0.05 µA w/*smaRTClock* off
  - Operating Range: -40 to 85°C

- **External Memory (IC2):** Microchip 24FC1025 I^2C Serial EEPROM (128 KB) [24]
  - Clock Frequency (max): 1 MHz (400 kHz for VCC < 2.5V)
  - Write Mode: 128-byte pages, 5 ms write cycle time (max)
  - Supply Voltage: 1.8 - 5.5V
  - Standby Current: 100 nA typical at 5.5V (1 μA max over temp range)
  - Supply Current (max): 400 μA at 5.5V (read), 3 mA at 5.5V (write)
  - Operating Range: -40 to 85°C
- **Ambient Light Sensor (IC3):** AMS TSL13T Light-To-Voltage Converter [26]
  - Spectral Responsivity: 350-1020 nm (25%), 775nm (peak)
  - Turn-On Time (0-100%): 40 μs (typical)
  - Rise Time (10-90%): 7.2 μs (typical)
  - Fall Time (10-90%): 6.8 μs (typical)
  - Supply Voltage: 2.7 - 5.5V
  - Supply Current: 660 μA at 3.0V (typical), 1.0 mA (max)
  - Operating Range: 0 to 70°C
- **Accelerometer (IC4):** Analog Devices ADXL345 3-axis Digital Accelerometer [25]
  - Dual Operation: simultaneous operation as accelerometer and tilt-sensor
  - I^2C and SPI command and data digital interfaces
  - 32-level sample FIFO
  - Built-in motion detection functions
  - Configurable interrupt modes mappable to two interrupt pins
  - Acceleration Range: Configurable to $\pm$ 2/4/8/16*g*
  - Sensitivity: 4 mg/LSB in all *g*-ranges (13-bits for $\pm$ 16*g*)
  - Shock Survivability: 10,000*g* (min)
  - Bandwidth: Configurable (max 3200 Hz sample rate)
  - Turn-On Time: 1.4 ms (typical)
  - Supply Voltage: 2.0 - 3.6V
  - Supply Current: 130 μA at 2.5V (typical)
  - Operating Range: -40 to 85°C
- **Battery:** Panasonic CR2016 (110 mAh), CR2025 (165 mAh), CR2032 (220 mAh) Battery [28]
  - 3V Lithium Coin Cell
  - Operating Range: -30 to 80°C

2.3 *SenseModule* Schematic Diagram

The schematic diagram for the *SenseModule* used to collect the data presented later in this paper is given in Figure 2 below. The schematic was created using V5.0 of CadSoft's Eagle Schematic and PCB Layout software [19].

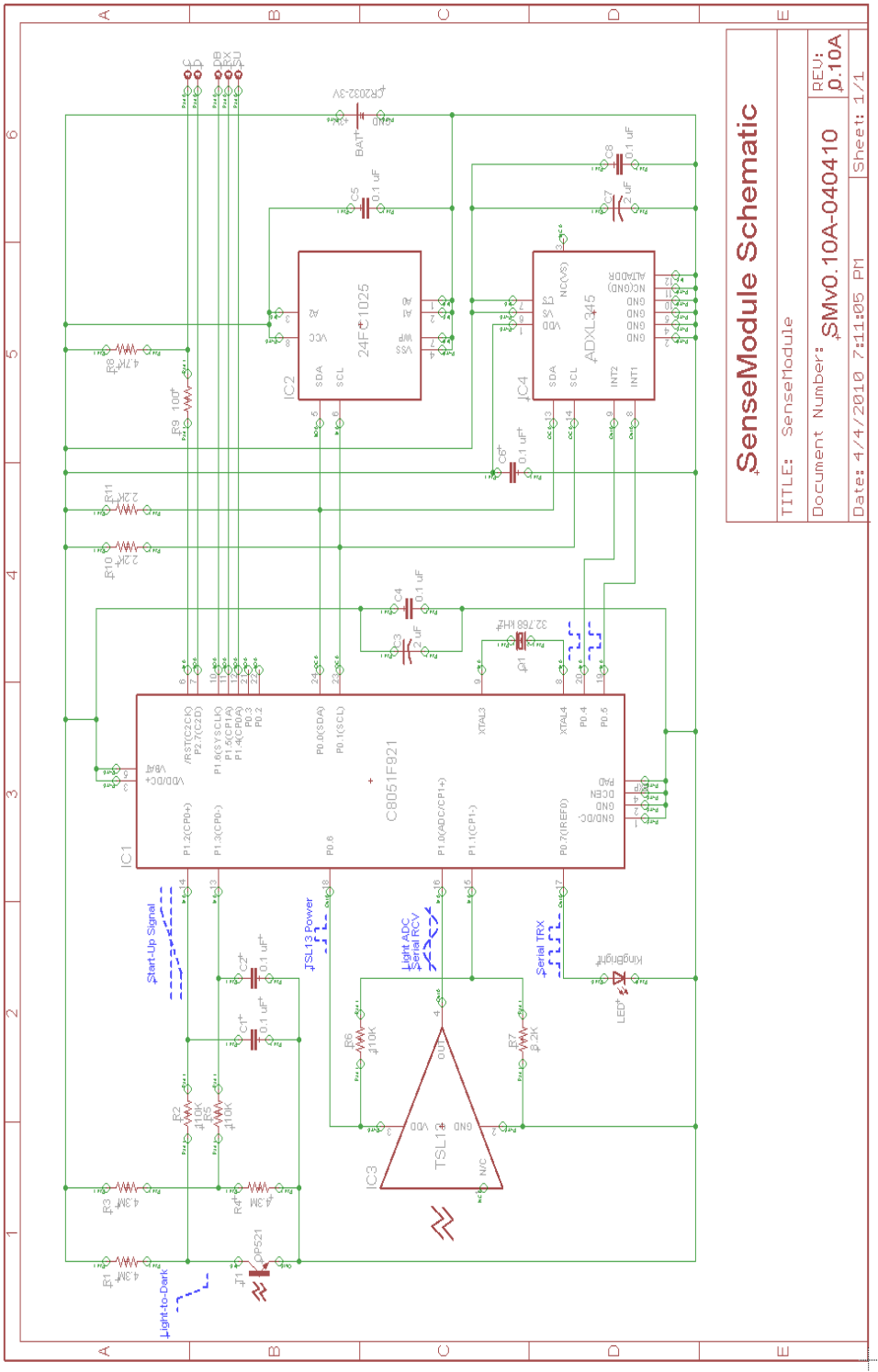

Figure 2: *SenseModule* Schematic Diagram

2.4 *SenseModule* PCB Layout

The *SenseModule* printed circuit board layouts in Figure 3 and Figure 4 below were created from the schematic shown in Figure 2. The PCBs were laid out using V5.0 of CadSoft's Eagle Schematic and PCB Layout software [19].

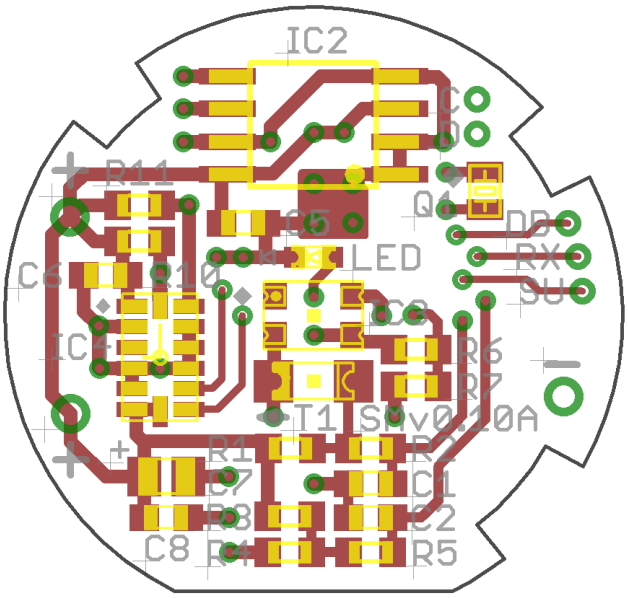

Figure 3: *SenseModule* PCB Layout (Top)

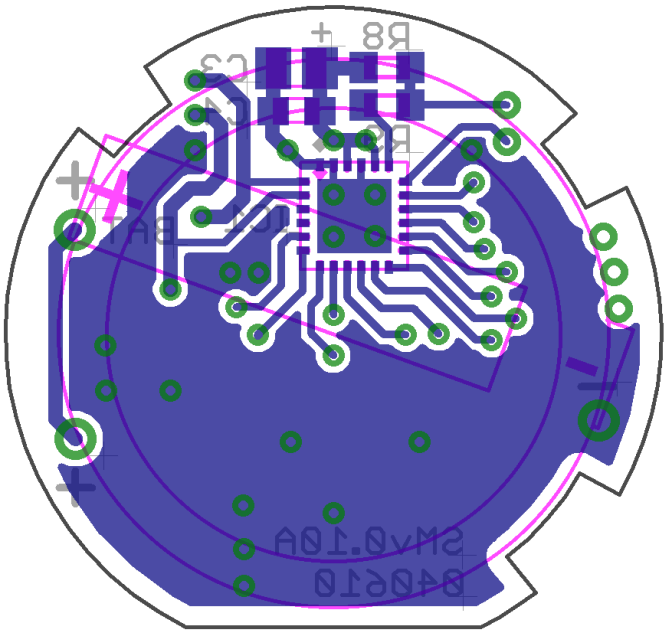

Figure 4: *SenseModule* PCB Layout (Bottom)

2.5 *SenseModule* Theory of Operation

The *SenseModule* schematic is divided into five distinct sub-circuits:

- Start-up circuit
- Microprocessor (μP) and memory circuit
- Ambient light sensor circuit (doubles as receiver)
- Accelerometer circuit
- Transmitter circuit

The start-up circuit is always active and uses one of the microprocessor's on-board comparators (CP0) to constantly monitor the ambient light level for signs of user activity. Under the proper circumstances, the comparator issues a reset signal to the microprocessor, which causes the *SM* to wake up from standby mode. In order to conserve battery power, and minimize the size of the battery, the start-up circuit is designed to draw less than 2 μA at all times.

Although IC1 (8051F921 microprocessor), IC2 (24FC1025 EEPROM), and IC4 (ADXL345 3-axis accelerometer) are always powered, they are all in stopped or halted states that draw very little current while the *SM* is in standby mode. When the start-up circuit wakes the *SM*, the μP selectively supplies power to IC3 (TSL13 Light-to-Voltage Converter), and wakes up the ADXL345 circuit, as needed. For the same reasons as the start-up circuit, the entire *SM* is designed to draw minimal (< 3 μA) combined quiescent current while in standby mode. Since the *SM* is powered by a 3V lithium coin cell, the average current that the *SM* draws while running is limited to approximately 1 mA. That current specification imposes a limitation on the maximum system clock frequency.

The ambient light sensor circuit is selectively powered by the microprocessor. The ambient light sensor performs double duty: it senses the ambient light level while the *SM* is collecting sensor data; and it serves as the infrared communications receiver for *ComModule* transmissions.

The *SM* applies power to the accelerometer circuit only while the *SM* is collecting sensor data. The accelerometer is powered-off, otherwise. The accelerometer can sense the following events:

- Motion of the ball during the bowler's approach and release
- Tilt of the ball in all three dimensions with respect to gravity
- Impacts of the ball with the lane and the pins
- Linear and angular acceleration of the ball as it rolls down the lane.

The optical transmitter is only powered when the *SM* is responding to a *CM* command – generally while transferring sensor data to the *CM*. The transmitter software implements an inverted return-to-zero (iRTZ) serial UART protocol, in which the space state (logic 1) is dark, and the mark state (logic 0) is a light pulse of ≤ 50% duty cycle. The LED has been chosen to maximize light output at low current, and is driven by the μP's programmable constant current source, set for 500 μA LED current.

2.5.1 Start-Up Circuit

Requirements for the operation of the *SenseModule* include that it must be automatic and transparent to the bowler. The presence of the *SM* must not impinge on the bowler's normal "feel" for the ball. Further, operation of the *SM* while in its data collection mode must not alter the bowler's normal routine in any way.

The *SM* has low quiescent power consumption under all ambient lighting conditions. The start-up circuit is always powered, and must be available to start the microprocessor at any time, without any intentional interaction on the part of the bowler, except for placing a finger in the insert as part of their normal delivery. Furthermore, it cannot be assumed that the ball will be stored in a dark ambient condition, i.e., a bowling ball bag, locker, or closet. Thus, the circuit draws very low current whether phototransistor T1 is exposed to constant dark conditions, constant light conditions, periodic light-to-dark and dark-to-light transitions resulting from the normal rolling of the ball, or to the 120 Hz waveform that overhead fluorescent lights in a bowling establishment impose on the ambient background light.

The start-up circuit issues a start-up signal whenever the bowler places a finger in the insert, or when the *ComModule* is placed over the finger hole. The value of R1 and the ratio of R3 to R4 have been selected such that T1 is driven sufficiently conductive under ambient background conditions so that the act of placing a finger in the insert (or covering the finger hole with the *CM*) causes a light-to-dark transition. Assuming that some background light is present, either of those conditions will sufficiently reduce the ambient light reaching T1 to cause T1 to cease conducting.

The start-up circuit is designed to be insensitive to most nuisance light-to-dark transitions. There are many instances that can cause such transitions:

- Ball rolling over the finger hole on the ball return
- Ball rolling on an above-ground ball return
- Ball spinning on the ball return wheel in the pinsetting machine
- Ball being picked up without a finger placed in the insert
- Jostling of the ball while other bowling balls are being retrieved, or by other balls coming back to the ball return

The start-up circuit must also not issue a start-up signal when the module is already operational:

- While the ball is rolling down the lane
- During communications with the *ComModule*

The start-up circuit for the module consists of phototransistor T1 (Optek 521), resistors R1–R5, capacitors C1 and C2, and the μP's on-board CP0 comparator. When CP0 is configured as part of the start-up circuit, its inputs and asynchronous output (CP0A) are assigned to the following μP port pins:

- CP0+ (positive input): P1.2
- CP0- (negative input): P1.3
- CP0A (asynchronous output): P1.4

The voltage at the junction of R1 and T1 provides the raw start-up signal to IC1 at CP0+. R1 also limits the maximum current through T1 to ~ 0.7 μA. R3 and R4 form a voltage divider that sets the negative threshold level at CP0-, and draw ~ 0.3 μA. R1, R2, and C1 form an RC filter that sets the minimum dark pulse duration that can reach μP's CP0+ input. R3, R5, and C2 form a second matching RC filter for the μP's CP0- input to help limit differential noise at the CP0- input, while also providing a bias current that closely matches that of the CP0+ input.

The values for R1, R3, and R4 have been chosen to limit the maximum current to less than 2 μA under all lighting conditions. In addition, the values for the RC filters have been chosen to limit the response of CP0+ to dark pulse durations of greater than 500 ms. Since there is such a slow rise-time at CP0+, the CP0 comparator has been configured for start-up operation, as follows:

- Positive Hysteresis: 20 mV (max available)
- Negative Hysteresis: 20 mV (max available)
- Response Time: 5 ms (max available)

The output of the CP0 comparator has been configured to issue a hardware reset internal to the μP upon a positive transition of CP0's internal output. The asynchronous CP0A output is connected to port P1.4 so that the start-up operation can be monitored and evaluated on an oscilloscope.

2.5.1.1 Theory of Operation

In a steady-state light condition, T1 conducts, and the voltage at CP0+ is lower than that at CP0-, and the internal CP0 output is held low (logic 0), and does not trigger the internal μP reset.

In a steady-state dark condition, T1 is non-conducting, and the voltage at CP0+ is higher than that at CP0-. Thus, the internal CP0 output is held high (logic 1), and does not trigger the internal μP reset.

In a transition from light to a sufficient level of darkness, T1 stops conducting (its series resistance rises significantly) so that the voltage at R1 rises, slowly charging C1 through R1 and R2. With a prolonged duration of sufficient darkness, the voltage at CP0+ eventually rises above that at CP0-, and CP0A transitions high, creating a positive transition that causes a hardware reset, starting up the μP. Upon a HW reset, the CP0 configuration is also reset such that it no longer can issue a HW reset to the μP.

In transition from dark to sufficient brightness, T1 begins conducting (its series resistance falls dramatically), so that the voltage at CP0+ falls below that at CP0-, and CP0A transitions to a logic 0, presenting a negative transition that has no impact on the μP.

Further, T1 conducts under sufficient light conditions. When T1 is conducting, it is draining charge from C1 through R2. C1 only has the opportunity to charge to a voltage level above that set by the voltage divider formed by R3 and R4 when T1 is prevented from conducting for a sufficient amount of time. Thus, CP0 only responds to dark pulses of a minimum duration (set by R1, R2, and C1), while short transient pulses, as well as repetitive pulses of sufficient frequency are filtered out from detection.

A description of a typical start-up scenario of the *SenseModule* follows.

1) The bowler picks up the ball. The act of picking up the ball is sufficient to introduce enough light to the finger hole to cause T1 to conduct, which results in CP0 internally emitting a logic low. Recall that the μP does not respond to dark-to-light transitions.
2) In preparation for delivering the ball to the lane, the bowler places his fingers in the finger holes, which cuts off light to T1 for a duration sufficient to charge C1 (through R1 and R2) to a level above that set at CP0- by R3 and R4. This is a light-to-dark transition.
3) As a result, CP0 transitions from low to high, which triggers the internal HW reset for the μP.

4) While the μP is in reset, it disables CP0, and applies weak logic pull-up resistors to both CP0+ and CP0-, all of which prevent CP0 from issuing a subsequent HW reset until it is once again configured for that purpose (just before the *SM* returns to **SleepMode**).
5) The μP comes out of reset and vectors to its reset code, and commences program execution.
6) Finally, when the μP is ready to shut down, it again configures CP0, allows the CP0 output to settle, and then enables CP0 to again detect start-up pulses. The μP then immediately enters its internal micro-power **SleepMode**.
7) If (on the rare occasion) a valid start-up light-to-dark transition occurs and triggers CP0 before the μP has had the opportunity to enter **SleepMode**, the start-up circuit issues a valid reset pulse, and the μP is once again placed in reset, and then vectors to its reset code (returning to step 4).

2.5.2 Light Sensing Circuit (IC3 – TSL13)

The ambient light sensing circuit is based on the AMS Sensing Solutions TSL13 light-to-voltage converter chip. The TSL13 outputs a linear voltage in relation to the light intensity (irradiance) that falls upon it. The output is ratiometric to the supply voltage [26].

The TSL13 (IC3) is selectively powered by the μP to limit power consumption. The TSL13 is powered through VDD (pin 3) from P0.6 on the μP, which is configured as a digital push-pull output pin. The TSL13's ground pin (*GND*, pin 2) is connected to battery ground. The TSL13's output pin (*Out*, pin 4) connects to P1.0 of the μP, which is configured as an analog input pin, which the μP configures as an ADC input or as the CP1+ input, as necessary.

Resistors R6 and R7 form a voltage divider between the TSL13's VDD and GND pins, while the junction between R6 and R7 connects to P1.1 of the μP, which is configured as the CP1- input. The R6-R7 voltage divider sets the threshold voltage for infrared serial light-pulse detection.

The light sensing circuit serves dual purposes:

1) It senses the ambient light level impinging upon the sensor module during the bowler's approach, and while the ball is rolling down the lane. In this case, P1.0 of the μP (to which the output of the TSL13 is connected) is configured as an ADC input channel.
2) It detects optical-based serial data transmissions to the *SenseModule*. In this case, P1.0 of the μP is configured as the positive input (CP1+) to its onboard CP1 comparator.

2.5.2.1 Sensing Mode

When the *SenseModule* is sensing ambient light, the μP periodically measures the ambient light level impinging upon the module. The μP configures P1.0 as the ADC input for the TSL13, applies power to the TSL13 through P0.6 (outputting a logic 1 at P0.6), waits an appropriate amount of time for the TSL13 to stabilize (100 μs), and then initiates an ADC sample of the TSL13's output pin. After the μP's internal ADC circuitry signals that it has completed the sample conversion, the μP removes power from the TSL13 by setting P0.6 to logic 0.

The μP "sees" the moment the bowler releases the ball by detecting the dark-to-light transition as the bowler's finger leaves the finger insert. While the ball is rolling, the μP periodically samples the ambient light sensor output and stores those samples in the FC1025 EEPROM, using a method and hardware similar to that used to capture the light waveforms in the original *SMARTDOT* sensor module.

2.5.2.2 Communications Mode

While the sensor module is in communications mode, the μP provides constant power to the TSL13 through P0.6. The μP configures P1.0 as the positive input (CP1+) and P1.1 as the negative input (CP1-) for its onboard CP1 comparator. When the voltage from the TSL13 at the CP1+ pin exceeds that determined by the R6-R7 voltage divider connected to the CP1- pin, CP1 outputs a logic 1 internal to the μP. When the voltage at the CP1+ pin falls below the threshold set by R6-R7, CP1 outputs a logic 0 internal to the μP. The *SenseModule* embedded software then interprets the internal CP1 output to emulate a serial receiver.

The transmitter is a visible/infrared LED driven by the μP's programmable constant current source (IREF0), and is connected to the μP via P0.7. IREF0 is configured to drive the transmit LED with 500 μA. To further limit the LED current during transmission, an inverted return-to-zero (iRTZ) scheme is used for serial transmission, similar to the IRDA standard, where serial '0' bits will be transmitted as short light pulses (≤ 50% duty cycle), while serial '1' bits are represented by the interstitial dark times.

2.5.3 Accelerometer circuit (IC4 – ADXL345)

The accelerometer circuit is based on the Analog Devices ADXL345 3-axis accelerometer chip. From the ADI data sheet [25],

> "The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 *g*. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I^2C digital interface.
>
> The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 m*g*/LSB) enables measurement of inclination changes less than 1.0°."

The μP can place the ADXL345 (IC4) in a micro-power standby mode through commands issued via the I^2C interface, thus VDD (pin 1) of the ADXL345 is always connected to the battery. The hardware chip select ($\overline{CS}$) pin is pulled to a logic '1', while the analog supply (VS) is connected to the battery via a separate line than the digital logic. The ground pins (pins 2, 4, 5, 10, 11), plus ALTADDR (pin 12), are connected to battery ground.

All chip configuration and data retrieval operations are performed via the I^2C interface (SDA and SCL). The two configurable interrupt pins (INT1 and INT2) are connected to the μP's P0.5 and P0.4 port pins, respectively. Those port pins are then configured as external interrupts for the μP.

The ADXL345 is the primary sensor for the *SenseModule*. The ADXL345 measures both the static force due to gravity (3-axis tilt sensor), as well as the dynamic forces exerted upon the chip. Thus, the

ADXL345 can detect the orientation of the *SM* with respect to the vertical acceleration due to gravity, as well as the instantaneous centripetal acceleration generated by the angular velocity (rotation) of the ball, along with any impacts and vibration the ball experiences. Digital signal processing (DSP) techniques are then used to separate those components into their constituent parts.

The ADXL345 is situated on the *SM* PCB in such a way that the *X* and *Y* axes are oriented tangentially to the ball's surface, so that they respond primarily to the tilt of the module relative to gravity and less so to the centripetal acceleration of the ball. The *Z* axis is oriented in line with the radius of the ball, and responds more strongly to the centripetal acceleration than the *X* and *Y* axes. The ADXL345 provides the raw 3-axis accelerometer data necessary for determining the motion of the ball during the bowler's approach and delivery, the forces the bowler applies to the ball during the release motion, and the reaction of the ball as it rolls down the lane. It also detects the ball's impact with the lane, and ultimately with the pins.

The μP configures the ADXL345 to run autonomously, collect 3-axis accelerometer samples at 200 HZ, store those samples in its internal sample buffer, and issue an interrupt when the sample buffer reaches 25 samples. Thus, the μP can be put into a low-power idle mode while the ADXL345 fills up its sample buffer. With the ADXL345 configured as described, it nominally draws 350 μA while collecting samples, and much less than 1 μA while in standby mode.

2.5.4 Microprocessor (IC1 – 8051F921)

The 8051F921 microprocessor (μP) is always powered, but spends most of its time in a micro-power sleep mode. While in sleep mode, only two functions are powered: the CP0 comparator, which is connected to the start-up circuit, and the *smaRTClock*, which keeps track of the date and time-of-day. CP0 is configured to issue a reset to the μP, which causes the μP to execute its internal self-configuration routine before beginning program execution at the reset vector. At that point, the embedded software takes over and further configures the μP as needed for the *SenseModule* application. After the *SM* has concluded processing the event that woke it up, it shuts down all unnecessary functions and returns to sleep mode, awaiting the next wake-up event.

The 8051F921 is a highly configurable system-on-a-chip. The basic configuration parameters for the various microprocessor functions that the *SM* utilizes are presented below [23].

2.5.5 System Clock

The 8051F921 μP has an onboard system oscillator, which is factory programmed for 24.5 MHz ± 2%. The system clock can be configured for slower clock rates (divide by 2, 4, or 8). Since the *SenseModule* application requires low power consumption, the system clock is configured for the minimum clock rate: 24.5 MHz / 8 = 3.05 MHz. In addition, the clock rate can be changed on-the-fly. If additional processing speed is necessary, the system clock frequency can be temporarily increased, as needed.

2.5.6 Low Power Modes

The μP has several low-power modes that can be used to significantly reduce power consumption. The μP's internal sleep mode allows it to remain powered, while shutting off current to all peripherals except for the low-power comparators and the smaRTClock. In that configuration, the μP draws ~1 μA.

The µP can resume processing from sleep mode based a number of configurable events. In this application, those events are the CP0 comparator output, and the *smaRTClock* functions.

The µP also has an internal idle mode that allows the timers, comparators, and ADC to run, while shutting off any unnecessary circuitry. The µP can resume processing from idle mode, based on any interrupt. Idle mode is used to significantly reduce current consumption during sampling.

2.5.7 Real Time Clock (RTC)

The µP has an on-board micro-power real time clock (*smaRTClock*) that is used to track chronological time, as well as provide correlated time-stamps for the ambient light and acceleration samples. The RTC function is always enabled, and allows the *SenseModule* to keep track of the date and time-of-day, even when the *SM* is in sleep mode. A 32.768 kHz crystal oscillator is used to drive the *smaRTClock*.

2.5.8 Port Pins

Another key feature of the µP is the ability to assign selected peripheral functions to specific port pins, as needed. This capability facilitates optimal use of port pins, as well as helps optimize PCB layout.

2.5.9 Comparators

The µP has two highly-configurable, on-board, low-power analog comparators (CP0 and CP1). Comparator CP0 serves as part of the start-up detection circuit, while CP1 is the serial reception edge detector for the software UART.

2.5.10 Analog-to-Digital Converter (ADC0)

The µP has a highly configurable, on-board 300 Ksps 10-bit analog-to-digital converter (ADC0) that accepts signals from a 13-channel analog multiplexer. ADC0 is used to sample the ambient light waveform at the output of the TSL13 light-to-voltage converter.

2.5.11 I²C Bus

There is byte-wide I^2C bus function built into the µP. The I^2C bus function is used to interface with the ADXL345 accelerometer and the 24FC1025 serial EEPROM.

2.5.12 Timers

There are 4 timers available for assignment to a variety of functions. The I^2C bus requires a timer, as does the ambient light sampling timer and the ADC0 converter. In addition, the serial communications interface utilizes a timer, although serial communications never run in conjunction with sampling.

2.5.13 Interrupts

The µP has an extensive two-level prioritized interrupt system. The UART bit-slice timer, the ADC0 sample timer and ADC0 conversions, the I^2C bus, the comparators, the Port 0 pins connected to the ADXL345 interrupt pins, and the *smaRTClock* all issue interrupts that are utilized in this application.

2.6 EEPROM (IC2 – 24FC1025)

The 24FC1025 EEPROM has a capacity of 128 Kbytes, arranged as 1024 pages containing 128 bytes a page. The EEPROM has a 128-byte write buffer that allows full page writes to be received in one I^2C transaction, and committed to memory in a single write cycle. It also has a maximum write time of 5

ms, for any single write transaction, including writing an entire 128-byte page. As with the μP, the 24FC1025 I^2C serial EEPROM chip is always powered, but draws negligible current when not being accessed and not committing data to memory from its write buffer [24].

Although the page writing capability is 5 ms, it takes longer than that (~7 ms) to transfer 128 bytes to the EEPROM's write buffer via the I^2C bus. Thus, it takes ~12 ms to transfer and write a page to EEPROM, and ~7 ms to read a page from EEPROM (or the ADXL345). The page writing capability of the EEPROM is heavily leveraged to take advantage of the faster overall data transfer rates, as well as to limit the amount of time the EEPROM spends committing data to memory. The EEPROM draws ~250 μA during reads and ~2 mA while committing data to memory. During sampling, the *SenseModule* writes 9 sample pages to EEPROM per second. By limiting the number of individual reads and writes to just those 9 pages, the active duty cycle of the EEPROM has been reduced to about 10%, resulting in an average current draw of about 115 μA, while sampling.

The EEPROM goes "silent" when it is in the process of committing its write buffer to memory. The *SM* uses that characteristic to find out when the EEPROM is ready to process the next transaction. If the EEPROM does not respond to a request, the *SM* software has a retry function that it executes the next time through its event processing loop. Thus, during sampling, there is no waiting on the EEPROM to finish committing a page to memory.

Section III: *SenseModule* Embedded Software

The *SenseModule* operation is intended to be fully autonomous. The *SM* should automatically turn itself on and off, without requiring the bowler to alter their normal routine in any way. Whenever the bowler rolls the ball down the lane, the *SM* should automatically record what it "sees" through its ambient light sensor (TSL13), and what it "feels" through its 3-axis acceleration sensor (ADXL345). Sometime after the *SM* has recorded the data, the bowler places the *ComModule* over the finger hole that holds the *SM*, at which point the *SM* should automatically detect the presence of the *CM*, and upload the recorded sensor data to the *CM*.

3.1 *SenseModule* Use Cases

Given the above requirements, there are three basic scenarios (use cases) to which the *SenseModule* must respond:

1) **Recording sensor data**: The bowler picks up the ball from the ball return, and places their fingers in the ball, which automatically wakes up the *SenseModule* from **SleepMode**. The *SM* detects a recordable event, and records sensor readings, from the time the bowler starts their approach, releases the ball, and delivers it to the lane, through the time the ball takes to traverse the lane, hit the pins, and fall into the pit at the back end of the lane. At that point, the *SM* automatically shuts down, and returns to **SleepMode**.
2) **Uploading sensor data:** At some later time, the bowler uploads the collected sensor data to the *ComModule* by placing the *CM* over the finger hole in which the *SM* is located. The *SM* and *CM* automatically detect each other's presence, and the *SM* switches to **CommandMode**. The *CM* issues a sequence of commands in order to retrieve the sensor data from the *SM*, and the *SM* uploads the sensor data under control of the *CM*. The *SM* automatically shuts down at the end of the command sequence, returning to **SleepMode**.
3) **Rejecting false wakeup conditions:** The bowler rolls the ball down the lane, and it reaches the end of the lane and falls into the pit. The *SM* automatically returns to **SleepMode** after having recorded its sensor data. The pinsetter picks up the ball and sends it back to the ball return. The ball emerges from the pinsetter into the light, just before rolling down the return ramp into the "subway." The light-to-dark transition that results is sufficient to wake the *SM*. Further complicating matters, the ball is rolling, appearing to the *SM* as if the bowler has again rolled the ball. The *SM* should automatically reject this event, and any other wake-up event that is not the result of the first two use cases.

3.2 Software Requirements

The above use cases impose a certain minimum set of required features and functions upon the *SenseModule* embedded software. There are additional requirements that must also be incorporated in order to make the *REVMETRIX* system easy and convenient to use.

3.2.1 Module Configuration

Since the *SenseModule* is still in development, optimal values for various light levels, release and shutdown detection algorithm parameters, time out values, etc. have yet to be determined. It is not yet

known whether a single set of parameters will be applicable across the various bowling styles. Thus, many of the parameters used by the *SM* should be configurable – stored in, and retrieved from EEPROM, as needed.

In addition, there is also a need for a unique module identifier, as well as a method for associating a particular *SM* with the ball in which it is installed. Password protection should also be implemented to discourage theft of the device and/or the bowling ball in which the *SM* is installed.

3.2.2 Power Management

The *SenseModule* has severe constraints on its battery capacity. The *SM* software should manage its internal and external hardware resources, as well as its overall run time to minimize current consumption. Such a strategy will both limit the required capacity of the battery (thus its size and weight), as well as extend the interval between battery replacements. Implementing that strategy involves a multifold approach:

1) Limit the system clock speed, and the time the μP spends operating at that clock speed.
2) Maximize the time the μP spends in low-power modes, such as **SleepMode** and **IdleMode**.
3) Selectively enable internal hardware functions (timers, ADC, I^2C bus) only when needed.
4) Selectively enable external peripherals (ADXL345, TSL13) only when needed.
5) Limit transactions on the I^2C bus: reads of the ADXL345, and reads/writes of the EEPROM.
6) Use time out intervals when waiting for events, e.g., valid wake-up, release, sampling, etc.
7) Limit the LED transmission current during serial communication.

3.2.3 Time Measurement

The *SenseModule* has several requirements for time measurement. It must have a stable MHz-level system clock. It must also sample the ambient light level (TSL13 output) at regular intervals. Even though the ADXL345 provides its own sample clock, the *SM* must be able to provide correlated time stamps between the light samples and the acceleration samples. The *SM* must also be able to keep track of chronological time so that the bowler can later associate the data captured in the *SM* memory with the dates and places of the captured data.

The *SM* uses a 3.05 MHz system clock derived from the μP's on-board 24.5 MHz oscillator. That system clock is accurate enough for clocking the μP and the various internal timers that are used for light sampling, serial communications, and the I^2C bus timing. The μP also provides an internal *smaRTClock* function that is used to implement a micro-power real time clock (RTC) when driven by a 32.768 kHz watch crystal. The RTC has a greater overall accuracy than the built-in system clock oscillator, so it is also used to provide a correlated time-stamp for the light and acceleration readings.

3.2.4 Ball Record Database (EEPROM)

It would be highly inconvenient and intrusive upon the bowler's normal routine if they were required to upload the sensor data after every roll of the ball. Thus, the *SenseModule* must be able to store sensor data resulting from multiple rolls of the ball. At a minimum, the bowler should not have to upload data from the *SM* more frequently than once per game.

3.2.5 Command Processing

The *SenseModule* uploads the data it collects to the *ComModule* via a serial infrared UART. A command protocol and series of commands are required that allow the *CM* to configure the *SM*, as well as interrogate it, and retrieve data from it. Since all non-volatile configuration and sensor data is stored in the *SM*'s EEPROM, two commands are required, at a minimum:

1) Write EEPROM Page
2) Read EEPROM Page

With the above two commands, the *CM* could read or write any combination of bytes in the *SM* by reading a page, modifying only the necessary bytes in that page, and then writing the modified page back to EEPROM. Other commands could be provided for the ease of interacting with the *SM*.

3.2.6 Infrared Serial UART (iRTZ Format)

The *SenseModule* communicates with the *ComModule* via an infrared serial interface using an inverted Return-To-Zero (iRTZ) format in order to limit the LED transmission current. Although the µP has an on-board serial UART, it is not configurable for the required iRTZ format. Thus, the iRTZ UART must be implemented in software.

3.2.7 Sensor Sampling

The main function of the *SenseModule* is to capture and store the raw data waveforms generated by the light sensor (TSL13) and the 3-axis accelerometer (ADLX345). Sufficient sampling rates are required that allow accurate reproduction of the captured waveforms, while still keeping the sample rate at a minimum in order to maximize the number of individual frames that the *SM* can store in its EEPROM, by limiting the amount of data collected per frame.

Due to the interrupt-driven nature of the sampling process, there are several data buffering techniques that should be utilized. Interrupts should be short-lived (quick), implementing just enough processing to retrieve the sample and place it in a buffer for event processing after return from the interrupt.

3.2.8 Sample Storage

3.2.8.1 Light Samples Buffer

Light samples are collected at a 240 Hz rate, and then averaged together to create a single 120 Hz reading. This averaging schemes filters out the 120 Hz "noise" imposed on the light waveform by the overhead fluorescent lighting found in most bowling establishments. To limit the amount of time that the µP is awake to collect and process light samples, the ADC0 interrupt should buffer a certain quantity of 120 Hz light samples before notifying event processing that new light samples are available.

3.2.8.2 ADXL Sample Buffer

A single 3-axis sample collected by the ADXL345 consists of 6 total bytes, 13 bits per axis, left-justified into a 16-bit word for each axis. The SMBUS0 interrupt reads the ADXL345 via the I^2C connection, and should assemble and store the 6-byte sample in its own buffer, until such time as the complete sample has been read, at which point it should copy the contents to a separate buffer for event processing to

handle. Thus, the interrupt can return to receiving the next sample without the possibility of overwriting the previous sample before event processing has a chance to execute.

3.2.8.3 Light Page Circular Buffer and ADXL Page Circular Buffer

During the bowler's approach, while the *SenseModule* is waiting for release, it must capture and store sensor data while it is also detecting the release condition from that sensor data. Only the most recent few seconds of data immediately preceding the release event must be stored in EEPROM; the rest can be discarded. The best way to do this is to implement circular buffers for both the light and accelerometer readings.

The interrupts capture and buffer the data, and event processing periodically transfers the buffer contents to the appropriate circular buffers. When the *SM* detects the release event, it can then start writing to the EEPROM from the current contents of the circular buffers.

3.2.9 Wakeup Validation

Whenever the *SenseModule* wakes up, it must first discriminate between the three given use cases, and determine which of them to follow. The *SM*'s first task is to detect a valid wake-up condition, and then detect the presence (or absence) of the *ComModule*, before moving on to considering recording sensor data.

A valid wake-up condition only occurs under extended dark conditions – either the bowler's finger is in the ball, or the *CM* is covering the finger hole. Thus if the *SM* "sees" too much light to soon after waking up, it should reject the wake-up condition as invalid, and immediately return to **SleepMode**.

Communication with the *CM* requires a very dark background light level. While the *SM* is validating the wake-up light level, it should also check for a level conducive to communication with the *CM*. If the light level immediately after waking up is sufficiently dark for communication, then the *SM* should first attempt to initiate contact with the *CM*. Only after contact with the *CM* cannot be established, should the *SM* proceed to sampling and recording data.

3.2.10 Approach and Release Detection

The *SenseModule* should not only capture the bowler's release of the ball and what follows afterwards, it should capture the motion of the ball during the bowler's approach leading up to release. However, only the last several seconds immediately preceding release are of interest. Thus, the *SM* should have a "pre-trigger" function that constantly samples the light and ADXL waveforms leading up to release (the "trigger"), but only retain the most recent few seconds of captured sensor data.

A release detection algorithm is required that can detect the bowler's release of the ball, and discern between an actual release event, and similar looking events, such as the ball emerging from the pinsetter on it is way to the ball return, and the ball emerging from the ball return itself. Both of those instances involve a dark-to-light transition, coincident with rotation of the ball.

In addition, there are wake-up events where the release condition will not occur. The *SM* should have a time out function associated with release detection, to limit the time it remains awake, waiting for

release. If the time out expires before a release condition is detected, the *SM* should automatically shut down and return to **SleepMode**.

3.2.11 Shutdown Detection

The *SenseModule* has severe constraints on data storage, as well as battery capacity. The *SM* should record data only up to the point that the ball has passed through the pins and fallen into the pit. Afterwards, it should automatically shut down operation and return to **SleepMode**.

A shutdown detection algorithm is required that quickly and reliably detects when the ball has ceased rolling. This routine should also be robust enough to discern the difference between the ball rolling down the lane, and the ball rotating while it is being sent back to the ball return.

As a "fail-safe" for shutdown detection, there should also be a shutdown time out function that halts sampling after a specified maximum time spent sampling data has expired.

3.3 EEPROM Memory Map

The *SenseModule* accumulates its sensor readings in external EEPROM. The EEPROM contents are divided into 128-byte pages (which is also the buffer size for write operations to the EEPROM). The EEPROM capacity is 1024 pages of 128 bytes each, for a total of 128 Kbytes. For this application, all reads from and writes to EEPROM are conducted a page (128 bytes) at a time, starting on an EEPROM page boundary.

The EEPROM is logically divided into three sections (3.3.1):

1) Configuration Page (page 0)
2) Ball Pointer Page (page 1)
3) Ball Record Array (pages 2 – 1023)

Page 0 of the EEPROM is the *SM*'s Configuration Page, which holds all of the configuration data and operational and system parameters for the *SM* (3.3.2).

Page 1 of the EEPROM is the Ball Pointer Page, which consists of an array of pointers to the first page of each Ball Record stored in EEPROM (3.3.3).

Pages 2 through 1023 hold the Ball Record array. Each Ball Record consists of a Ball Page, and a variable length collection of Light Pages and ADXL Pages that contain the captured sensor data for that particular Ball Record (3.3.5). The pages for each Ball Record are stored contiguously, with the Ball Page first (3.3.6), followed by a mix of Light Pages (3.3.7) and ADXL Pages (3.3.8). The Ball Page holds the header information for the Ball record. It also doubles as the first Light Page, with the bytes remaining after the header information being filled with Light samples. The Light Pages are stored in chronological order, as are the ADXL Pages, but there is no guarantee that the entire collection of the Ball Page, the Light Pages, and the ADXL Pages are all stored in chronological order with respect to each other.

The Ball Record array located in the EEPROM is organized as one large circular buffer. It spans 1022 pages, from EEPROM page 2 to EEPROM page 1023. The pages of a new Ball Record are written

contiguously to the EEPROM, overwriting the existing contents. When EEPROM page 1023 is written, the Configuration Page (page 0), and the Ball Pointer Page (page 1) are skipped over, and the next available sample page is page 2.

Before the *SM* begins committing a new Ball Record to EEPROM, it first retrieves the location of the next Ball Pointer entry to use (*nextBall*) , and the location of the next available sample page in the Ball Record array (*nextBallPage*).

At the conclusion of committing a new Ball Record to EEPROM, the Ball Pointer page is updated with the location of the Ball Page from the new Ball Record, and *nextBall* and *nextBallPage* are updated in the Configuration Page to point to the next available Ball Pointer and Ball Page respectively. The *SM* conducts a scan of the Ball Pointer array to verify which Ball Records are still valid, and which ones had at least one page that was overwritten. Any such records are marked as deleted in the Ball Pointer array, thus maintaining the integrity of the Ball Record database. See Figure 5 for a graphical depiction of the EEPROM memory map.

3.3.1 EEPROM Layout

The EEPROM is divided into 1024 physical pages of 128 bytes each. It is possible to access individual bytes of the EEPROM. However, in this application, it is more efficient to access a page at a time. Each page written to EEPROM has a Page Type associated with it, as the first byte of the page. The *SenseModule* can quickly identify the format of each page it reads from EEPROM by the Page Type.

Table 1: EEPROM Map

EEPROM MAP (128 Kbytes – 1024 128 byte pages)		
Addr: 0:0000	Addr: 0:0080	Addr: 0:0100
CONFIGURATION PAGE	**BALL POINTER PAGE**	**BALL RECORD ARRAY**
(Section 3.3.2)	(Section 3.3.3)	14 min @ 9344 bytes 32 max @ 4096 bytes
1 page (128 bytes)	1 page (128 bytes)	14-32 Ball Records (1022 pages)

The Configuration Page and Ball Pointer Pages are described below. The Ball Record Array is arranged as a circular buffer that contains the variable length Ball Records (3.3.5) associated with the Ball Record Pointers (3.3.4).

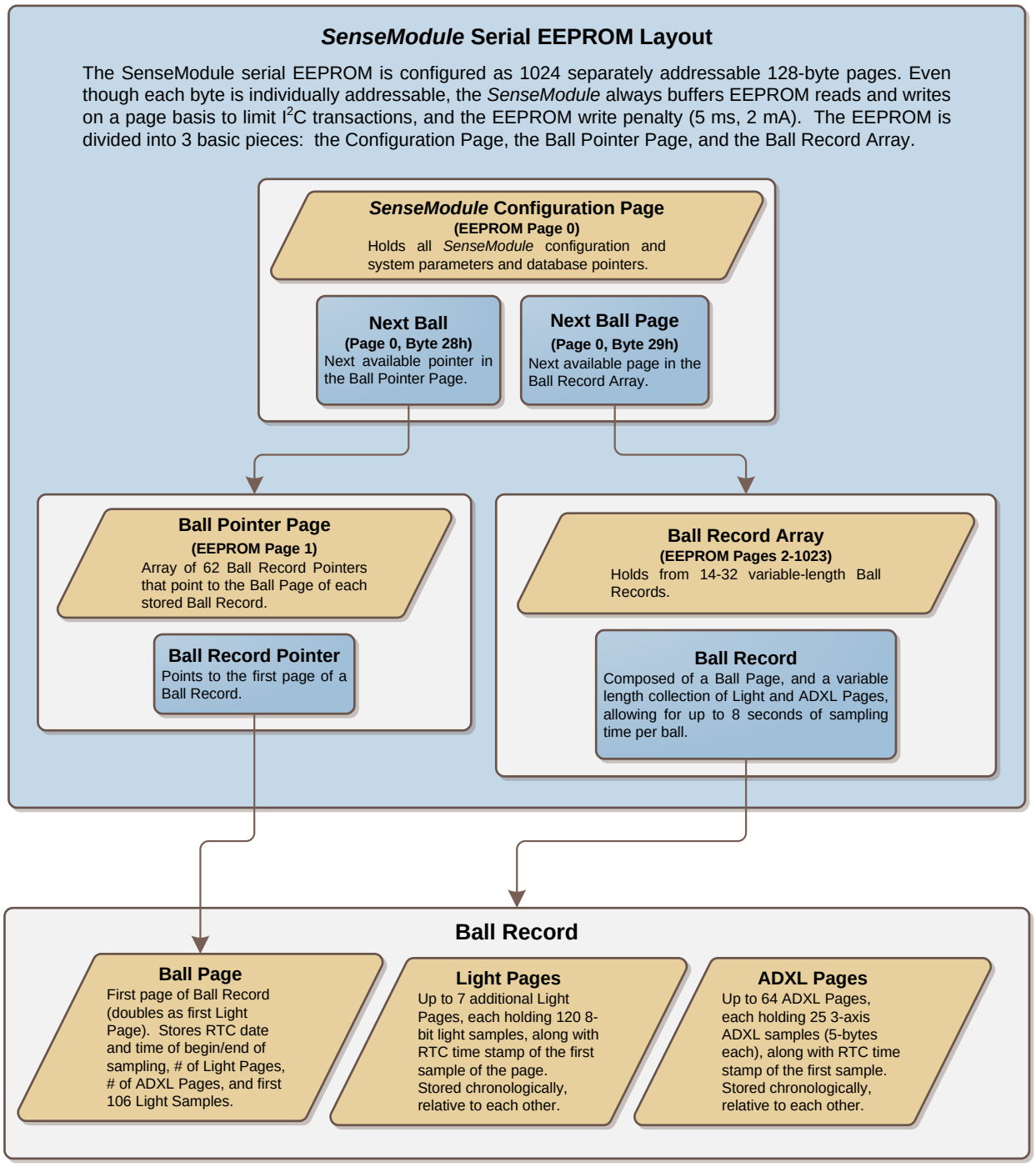

Figure 5: *SenseModule* Serial EEPROM Memory Map

3.3.2 Configuration Page

The Configuration Page holds the following configuration, system, and operational parameters. It is protected by a 16-bit CRC in order to detect corruption of the Configuration Page contents.

Table 2: Configuration Page Contents

PAGE 0	ADDRESS: 0:0000	CONFIGURATION PAGE (CONFIGPAGE)
0:0000	BYTE	Page Type: 0x20
0:0001	WORD	SW Version: loaded by firmware
0:0003	WORD	DB Version: loaded by firmware
0:0005	CHAR[10]	Ball Name: set by user
0:000F	WORD	Module ID: unique for this module (CRC from Ball Name)
0:0011	WORD	Module Password: supplied by user from *ComModule*
0:0013	DWORD	Time Base: set by *ComModule* after each command (time_t)
0:0017	WORD	Recording Mode: 0: Off (does not record, only responds to *ComModule*) 1: Single (records one Ball Record at a time) 2: Full (records until full, does not overwrite new Ball Records) 3: Automatic (always records – overwrites oldest Ball Records)
0:0019	BYTE	Light Buffer Pages: # of Light pages in Approach Buffer
0:001A	BYTE	ADXL Buffer Pages: # of ADXL pages in Approach Buffer
0:001B	BYTE	Max Sample Time: max # of seconds to sample
0:001C	BYTE	Light Release Threshold: min TSL13 values to detect release
0:001D	BYTE	ADXL Release Threshold: min ADXL Z-axis value to detect release
0:001E	BYTE	Light Sample Threshold: min Light value to continue sampling
0:001F	BYTE	ADXL Sample Threshold: min ADXL Z-axis value to continue sampling
0:0020	BYTE	ADXL Impact Threshold: min X, Y-axis values to detect impact
0:0021	WORD	Ball Count: total # of Ball Records created, rolls over at 65535 (16-bits)
0:0023	WORD	Deleted Ball Count: # of new Ball Records overwritten since count was last reset
0:0025	BYTE	New Ball Count: # of new Ball Records in DB
0:0026	BYTE	First New Ball: oldest new Ball Record in DB
0:0027	BYTE	Newest Ball: most recent new Ball Record in DB (last ball pointer used)
0:0028	BYTE	Next Ball: next available Ball Record in DB (next ball pointer to be used)
0:0029	WORD	Next Ball Page: next available Ball Page in DB (where next Ball Record is written)
0:002B	BYTE	Transmit Retry Count: # of retries during last download (for troubleshooting communications)
0:002C	WORD	Transmit Retry Page: first page that had to be retried (for troubleshooting communications)
0:002E	BYTE	Reset Reason: reason code for last EEPROM initialization
0:002F	WORD	System Clock adjustment: set by factory, and adjustable by *ComModule*
0:0031	WORD	Baud rate adjustment: set by factory, and adjustable by *ComModule*
0:0033	BYTE	RCV baud rate (28800, 57600, 1152000)
0:0034	BYTE	TRX baud rate (28800, 57600, 115200, 230400)
0:0035	BYTE	ADXL Sample Frequency (100 Hz, 200 Hz, 400 Hz, 800 Hz)
0:0036	BYTE	TSL13 Sample Frequency (120 Hz, 240 Hz)
0:0037	BYTE	Inactivity time: sleep if inactive for at least this long
0:0038	BYTE	Inactivity Threshold: (x-axis, y-axis, z-axis), register inactivity when all axes are below this threshold
0:0039	BYTE[69]	unused
0:007E	WORD	CRC (16 bits)

3.3.3 Ball Pointer Page

The Ball Pointer page contains an array of 62 pointers to Ball Records. Each Ball Record Pointer contains the EEPROM page address where the first page (Ball Page) of its corresponding Ball Record is located. Since Ball Records are variable length, and the Ball Record array is designed as a circular buffer, Ball Record Pointers are neither positional nor absolute, i.e., Ball Record Pointer 0 does not point to Ball Record 0. Rather, Ball Records are cumulative over the life of the *SenseModule*. After the Ball Record array is full, each new Ball Record overwrites the oldest Ball Record in the array due to the circular nature of the array. The Ball Record number is stored in the Ball Page of the record as Ball Count. The Configuration Page stores the locations of the oldest unread Ball Record, the newest Ball Record, and the next Ball Record Pointer and EEPROM page to use. The Ball Pointer Page is protected by a 16-bit CRC to detect corruption of the Ball Record Pointer array. Although the Ball Pointer Page holds 62 Ball Record Pointers, the current size of the EEPROM limits storage to a maximum of 32 Ball Records. A future incarnation of the *SM*, with an additional EEPROM chip, could hold up to 62 Ball Records.

Table 3: Ball Pointer Page Structure

BALL POINTER PAGE (EEPROM Page 1, Address: 0:0080, 128 bytes)			
0	1	2-125	126-127
PAGE TYPE	**UNUSED**	**BALL RECORD POINTER ARRAY (0-61)**	**PAGE CRC**
0x30		see below	16-bits
BYTE	BYTE	WORD[62]	WORD

3.3.4 Ball Record Pointer

Each Ball Record Pointer maintains the following status bits. The “In Use” bit indicates that the pointer currently points to a valid Ball Record. The “New” bit indicates that the Ball Record has not yet been uploaded. The “Deleted” bit indicates that the Ball Record associated with this pointer has been overwritten, and is no longer valid. The lower order 11 bits contain the EEPROM page address of the associated Ball Record.

Table 4: Ball Record Pointer Structure

BALL RECORD POINTER (2 bytes)					
15	14	13	12	11	10 - 0
BALL STATUS BITS					**BALL RECORD POINTER**
IN USE	**NEW**	**DELETED**	unused	unused	**EEPROM PAGE**
1: In Use 0: Available	1: New 0: Old	1: Deleted 0: Not Deleted			2 - 1023
WORD					

3.3.5 Ball Record

Ball Records are variable length, and consist of a Ball Page (as the first page), and a mix of up to 7 Light Pages, and up to 64 ADXL Pages, based on the maximum sampling time set in the Configuration Page. The Ball Page also doubles as the first Light Page. The Light Pages are stored in chronological order relative to each other, and the ADXL Pages are also stored in chronological order relative to each other, but there is no guarantee that the entire collection of Ball Page, Light Pages, and ADXL Pages are all stored in chronological order with respect to each other.

Table 5: Ball Record Structure

BALL RECORD (9216 bytes max: 72 sample pages * 128 byes)	
0	1-71 (max)
BALL PAGE	**SAMPLE PAGES**
Doubles as first Light Page (106 samples, 883 ms) (Section 3.3.6)	Mix of 1–7 Light Pages (Section 3.3.7), up to 7 seconds and 1-64 ADXL Pages (Section 3.3.8), up to 8 seconds
1 EEPROM page (128 byes)	Up to 71 EEPROM pages (9088 BYTES max)

3.3.6 Ball Page

The first page of a Ball Record is always a Ball Page. The Ball Page also doubles as the first Light Page, but has a lower sample capacity than the Light Pages, since it also holds the header data for the Ball Record.

Table 6: Ball Page Structure

BALL PAGE (128 bytes)									
0	1-4	5-6	7	8-11	12-15	16-19	20	21	22-127
BALL PAGE HEADER	**PAGE TIME STAMP**	**BALL COUNT**	**SAMPLE COUNT**	**BALL TIME STAMP**	**START TIME STAMP**	**END TIME STAMP**	**LIGHT PAGES**	**ADXL PAGES**	**LIGHT SAMPLES ARRAY**
see below	RTC time @ start of page		# of samples stored in page	RTC date @ start of sampling	RTC time @ start of sampling	RTC time @ end of sampling	# of Light pages in Ball Record (range: 1-8)	# of ADXL pages in Ball Record (range: 1-64)	8-bit Samples 0 – 105 (833 ms)
BYTE	DWORD	WORD	BYTE	DWORD	DWORD	DWORD	BYTE	BYTE	BYTE[106]

Each Ball Page Header points back to its Ball Record Pointer in the Ball Pointer Page.

Table 7: Ball Page Header Structure

BALL PAGE HEADER (byte 0 of Ball Page)							
7	6	5	4	3	2	1	0
PAGE TYPE BITS		**BALL RECORD #**					
1	1	0 - 61					
Ball Page Type = 11xxxxxxb		Ball index from Ball Pointer Page					
BYTE							

3.3.7 Light Page

Each Light Page can store up to 120 8-bit ambient light samples (1 second of data) collected from the TSL13 Light-to-Voltage converter. All Light Pages hold the full 120 samples, with the exception of the last Light Page, which may hold less, depending on when waveform sampling was terminated. Each Light Page also stores the RTC time stamp at the time the first sample was collected for that page.

Table 8: Light Page Structure

LIGHT PAGE (128 bytes)				
0	1-4	5-6	7	8-127
LIGHT PAGE HEADER	**PAGE TIME STAMP**	**BALL COUNT**	**SAMPLE COUNT**	**LIGHT SAMPLES ARRAY**
see below	RTC time @ start of page		# of samples stored in page	8-bit Samples 0 – 119 (1 second)
BYTE	DWORD	WORD	BYTE	BYTE[120]

Each Light Page Header points back to its Ball Record Pointer in the Ball Pointer Page.

Table 9: Light Page Header Structure

LIGHT PAGE HEADER							
7	6	5	4	3	2	1	0
PAGE TYPE BITS		**BALL RECORD #**					
1	0	0 - 61					
Light Page Type = 10xxxxxxb		Ball index from Ball Pointer Page					
BYTE							

3.3.8 ADXL Page

Each ADXL Page can store 25 5-byte compressed 3-axis acceleration readings (125 ms) collected from the ADXL345 3-axis accelerometer. All ADXL Pages hold the full 25 samples, since sampling always stops at the end of an ADXL Page. Each ADXL Page also stores the low-order word of the RTC time stamp at the time the first ADXL sample was collected for that page.

Table 10: ADXL Page Structure

ADXL PAGE (128 bytes)		
0	1-2	3 - 127
ADXL PAGE HEADER	**PAGE TIME STAMP**	**ADXL SAMPLES ARRAY**
see below	RTC time @ start of page (low-order WORD only)	Compressed 13-bit X,Y,Z-axis samples 0 – 24 (125 ms) see below
BYTE	WORD	ADXL Sample[25]

Each ADXL Page Header points back to its Ball Record Pointer in the Ball Pointer Page.

Table 11: ADXL Page Header Structure

ADXL PAGE HEADER							
7	6	5	4	3	2	1	0
PAGE TYPE BITS		**BALL RECORD #**					
0	1	0 – 61					
ADXL Page Type – 01xxxxxxb		Ball index from Ball Pointer Page					
BYTE							

3.3.9 ADXL Sample

The ADXL Samples that are stored in an ADXL Page are compressed versions of the 6-byte samples that are retrieved from the ADXL345. Each axis reading is 13-bits, originally left-justified into a 16-bit value. The most significant 13-bits of each axis reading are compressed into a 5-byte sample, before being copied to the ADXL Page. The *X*-, *Y*-, and *Z*-axis most significant bytes (MSBs) are kept intact so that those values can be easily isolated and tested during sampling. Their respective least significant bytes (LSBs) are compressed into the remaining 2 bytes, as shown. Essentially, the Z-axis LSB bits are allocated to the unused *X*-axis and *Y*-axis LSB bits, and then the original *Z*-axis LSB is discarded, which compresses the 6-byte sample into a 5-byte sample.

Table 12: ADXL Sample Structure (compressed)

ADXL SAMPLE (5 bytes - compressed)																	
0								1	2							3	4
***X*-AXIS (LSB), *Z*-AXIS (LSB)**								***X*-AXIS (MSB)**	***Y*-AXIS (LSB), *Z*-AXIS (LSB)**							***Y*-AXIS(MSB)**	***Z*-AXIS (MSB)**
7	6	5	4	3	2	1	0	bits 15-8	7	6	5	4	3	2	1	0	bits 15-8	bits 15-8
X-axis LSB bits 7-3					unused	*Z*-axis LSB bits 7-6			*Y*-axis LSB bits 7-3					*Z*-axis LSB bits 5-3				
BYTE								BYTE	BYTE							BYTE	BYTE

3.4 MainLoop

The *SenseModule* embedded software is structured around a variation of an interrupt and event-driven "super loop" architecture; **MainLoop** is the *SM*'s super loop process (see Figure 6). **MainLoop** progresses through a series of processing modes, directed by certain events, each of which is indicated by a unique event flag (EF). Several of the processing modes – specifically **CommandMode**, **ApproachMode**, and **SampleMode**, are each implemented as separate super loops. Each super loop executes its specific function until conditions encountered during repeated execution of the loop result in one or more EFs being set that eventually cause the loop to terminate, with control then returning to **MainLoop**.

The initial entry point into **MainLoop** is through the **ResetMode** process (see Figure 7) after a reset event occurs, but the normal execution path starts with the resumption of execution from within the **SleepMode** process (see Figure 8) following a wake-up event.

The *SM* spends the vast majority of its time in a micro-power sleep state which is entered and exited from within the **SleepMode** process. The *SM* awakens when ambient light is blocked from falling upon the *SM* for a sufficient period. After waking from sleep, *SM* execution proceeds to the **WakeUpMode** process (see Figure 9), which configures the μP following **SleepMode**, and then identifies the wake-up source.

If the wake-up source is valid (was initiated by the start-up circuit, and ambient light conditions are still sufficiently dark), then ***WakeUpEF*** is set, and execution proceeds to determine which of two operational branches to enter: sensor sampling, or command processing. If the wake-up source was not valid, then *SM* execution proceeds directly to the **CleanUpMode** process (see Figure 13), before returning to **SleepMode**.

If the ambient light is sufficiently dark for communications with *ComModule*, then the *SM* prompts the *CM*. If the *CM* responds, ***CommandReceivedEF*** is set after the prompt, and the *SM* proceeds to the **CommandMode** process (see Figure 10). The *SM* can repeatedly loop through **CommandMode**, sequentially processing a string of commands. When ***CommandReceivedEF*** is no longer set coming out of **CommandMode**, the *SM* proceeds to **CleanUpMode**, and then returns to **SleepMode**.

If the ambient light level condition was not sufficient for communication with the *CM*, or the *CM* did not respond to the prompt, the *SM* takes the sensor sampling branch, where the *SM* samples and stores the light and ADXL waveforms. The sensor sampling branch is composed of two processes: **ApproachMode** (see Figure 11) and **SampleMode** (see Figure 12), both of which are also super loops.

ApproachMode and **SampleMode** have similar functionality, with all data begin collected and stored in two circular buffers, one for ambient light samples, and one for ADXL samples. The major difference between the two modes is that in **ApproachMode**, no data is transferred to the EEPROM. Rather, the two sample buffers are used to implement a "pre-release" function, capturing the several seconds of sensor data that immediately precede release of the ball.

Upon a valid release of the ball (***ReleaseDetectedEF***), the *SM* immediately switches from **ApproachMode** to **SampleMode**. The *SM* continues to collect and store new sensor data in the circular buffers, but the *SM* also starts to commit (write) the sample pages currently stored in those buffers to EEPROM as part of the new Ball Record. Existing pages are written from the circular buffer tail, while new data is stored in the page currently at the head of the buffer.

The *SM* stops collecting sensor data when it detects that the ball has stopped rolling or the maximum number of light and ADXL pages have been collected. The *SM* exits **SampleMode** after it has committed the last of the collected sample pages to memory, and proceeds to **CleanUpMode**, before returning to **SleepMode**.

If a valid release was not detected during **ApproachMode**, the *SM* skips **SampleMode** and proceeds to **CleanUpMode**, before returning to **SleepMode**.

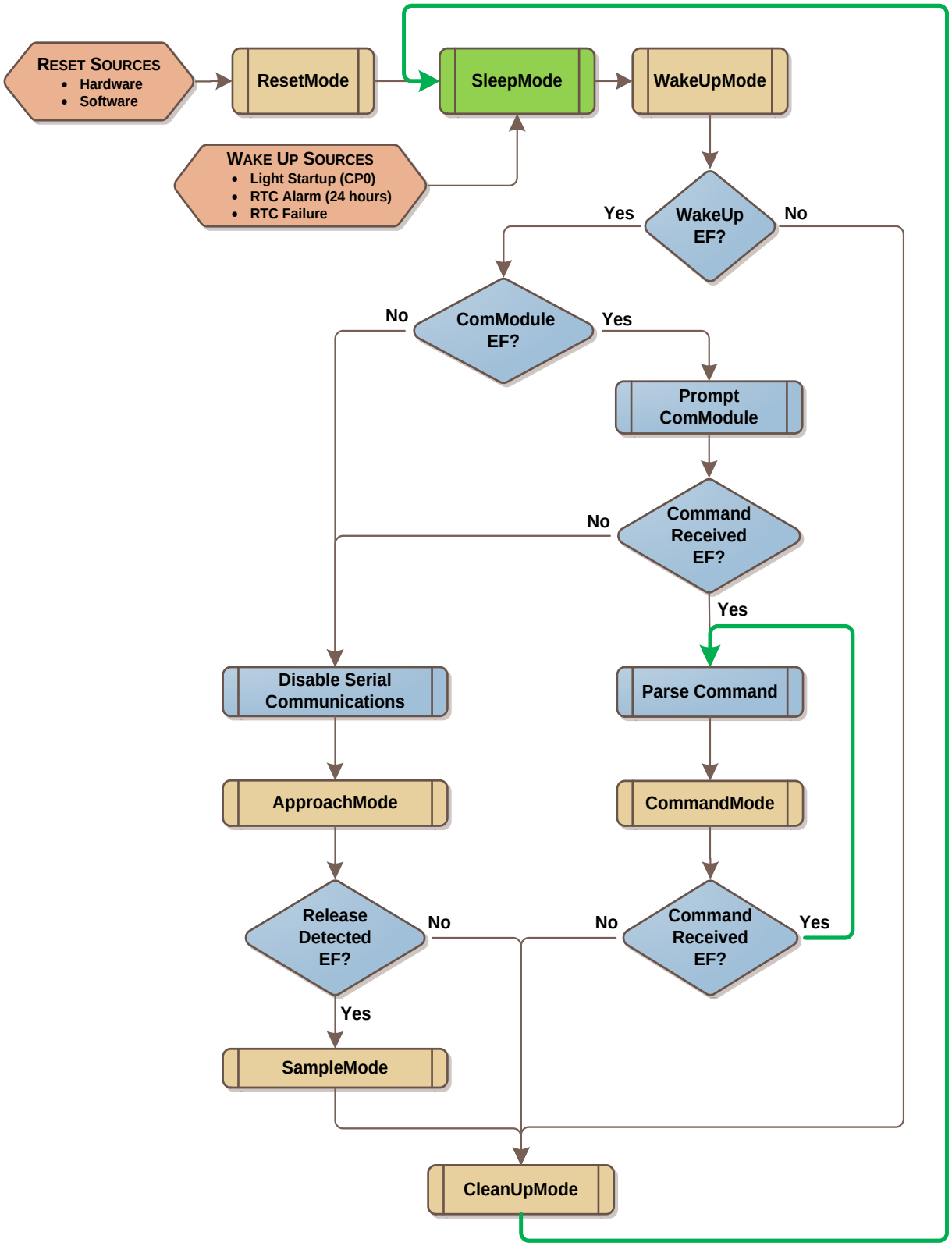

Figure 6: *SenseModule* MainLoop

3.5 ResetMode Process

Reset events can be triggered by hardware or software events, but occur rarely. Typical hardware reset events include loss-of-power detection, resulting from replacement of the battery, or from a watchdog timeout. Software reset events are issued as the result of built-in-test functions that detect anomalous conditions within the hardware, such as a CRC failure in the EEPROM, an RTC failure, or an unresponsive external peripheral, such as the ADXL345 chip.

Following any type of hardware or software reset event, the µP incorporates a fixed hardware delay, to allow its internal hardware functions to stabilize, including the system clock, and then starts execution at its reset vector address, which jumps to the **ResetMode** process (see Figure 7). **ResetMode** takes care of configuring the internal hardware functions (reset sources, watchdog, oscillators, *smaRTClock*), assigning port pin functionality, and finally enabling the interrupts.

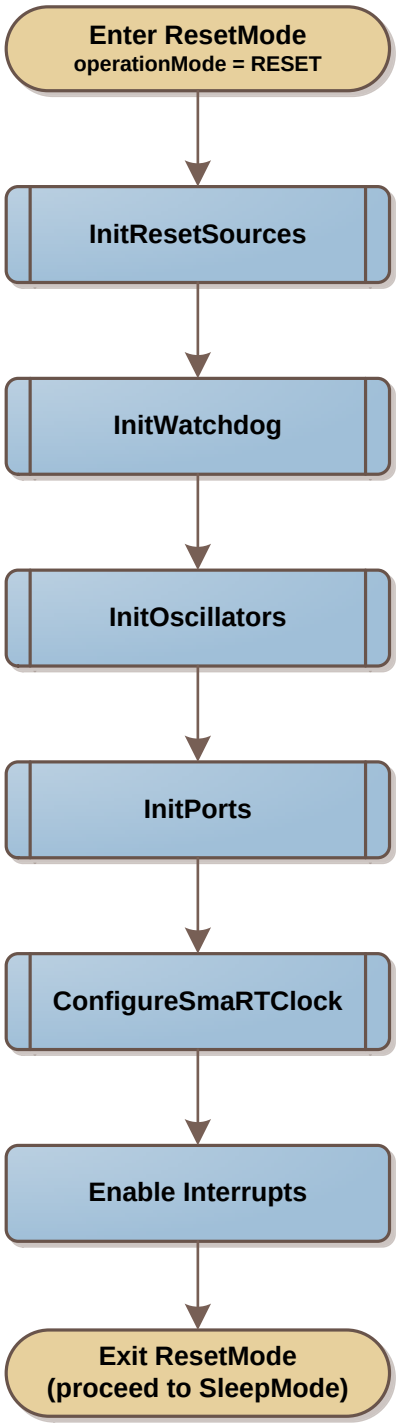

Figure 7: ResetMode Process

3.6 SleepMode Process

The *SenseModule* always returns to **SleepMode** (see Figure 8) at the end of every iteration through the main processing loop (as well as the rare occasion when it is coming out of **ResetMode**). It handles an orderly shutdown of the various internal μP hardware functions and external peripherals, and then places the μP in an internal micro-power sleep mode. The *SM* spends the vast majority of the time in **SleepMode**, waiting for a wake-up event to occur. While in **SleepMode**, only two internal hardware functions are enabled: the *smaRTClock* and CP0, which serves as the ambient light start-up source.

There are three events that can wake the μP from its internal sleep mode:

1) Ambient light start-up – a light-to-dark transition of sufficient length causes the μP's CP0 comparator to issue a wake-up event so that the *SenseModule* can start sampling.
2) Real Time Clock (RTC) 24-hour alarm – the *smaRTClock* reaches 24 hours, and issues a wake-up event so that the μP can update the RTC date.
3) RTC failure – the *smaRTClock* has failed, which issues a wake-up event so that the μP can reconfigure the *smaRTClock*, and hopefully recover from the failure.

Upon waking up, **SleepMode** determines the source of the wake-up event. If an ambient light wake-up event is detected, the *SM* exits **SleepMode**, and returns to **MainLoop**, which then calls **WakeUpMode**. Note that **SleepMode** does not enable any peripherals; **WakeUpMode** is responsible for that task.

If an RTC alarm awoke the *SM*, **SleepMode** processes the alarm event. If an ambient light wake-up event is not also present, **SleepMode** then puts the *SM* back to sleep.

If the source is from an RTC failure, **SleepMode** processes the event, and then issues a software reset.

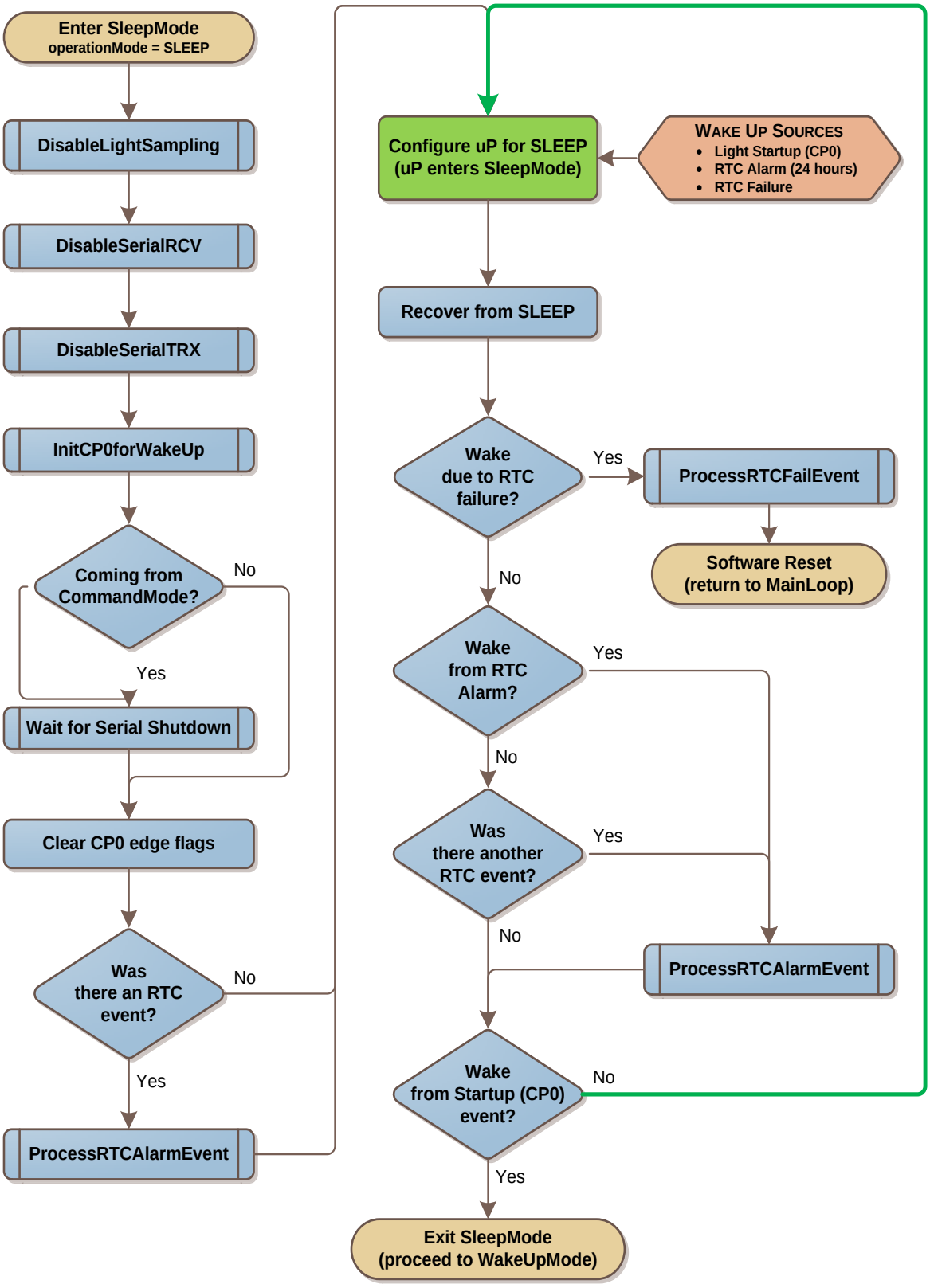

Figure 8: SleepMode Process

3.7 WakeUpMode Process

The *SenseModule* enters **WakeUpMode** (see Figure 9) upon detection of an ambient light start-up event coming out of **SleepMode**. **WakeUpMode** handles an orderly start-up of the μP and its peripherals, and then starts ambient light sampling. **WakeUpMode** samples the ambient light background to detect which type of start-up condition exists before passing execution back to **MainLoop**, which then decides how to proceed, based on the status of the two EFs (***ComModuleEF***, ***WakeUpEF***) that **WakeUpMode** can set.

In order for ***ComModuleEF*** to get set, the ambient light level must remain below the *ComModule* threshold for a minimum # of sample times, within the WakeUp time out period.

Similarly, in order for ***WakeUpEF*** to get set, the ambient light level must remain below the WakeUp threshold for a minimum # of sample times, within the WakeUp time out period.

The *CM* threshold is lower (darker) than the WakeUp threshold – thus if ***ComModuleEF*** is set, then ***WakeUpEF*** is set, as well. Ambient light sampling is halted when either EF is set, or when the WakeUp timeout is reached. If ***WakeUpEF*** has been set, **WakeUpMode** checks the integrity of the EEPROM database, and fetches the ConfigPage from EEPROM and places it into μP RAM.

Processing then returns to **MainLoop**, which then decides whether to proceed with execution of the loop (***WakeUpEF*** is set), or return the *SM* to **SleepMode** (***WakeUpEF*** did not get set, which is the result of an invalid start-up condition).

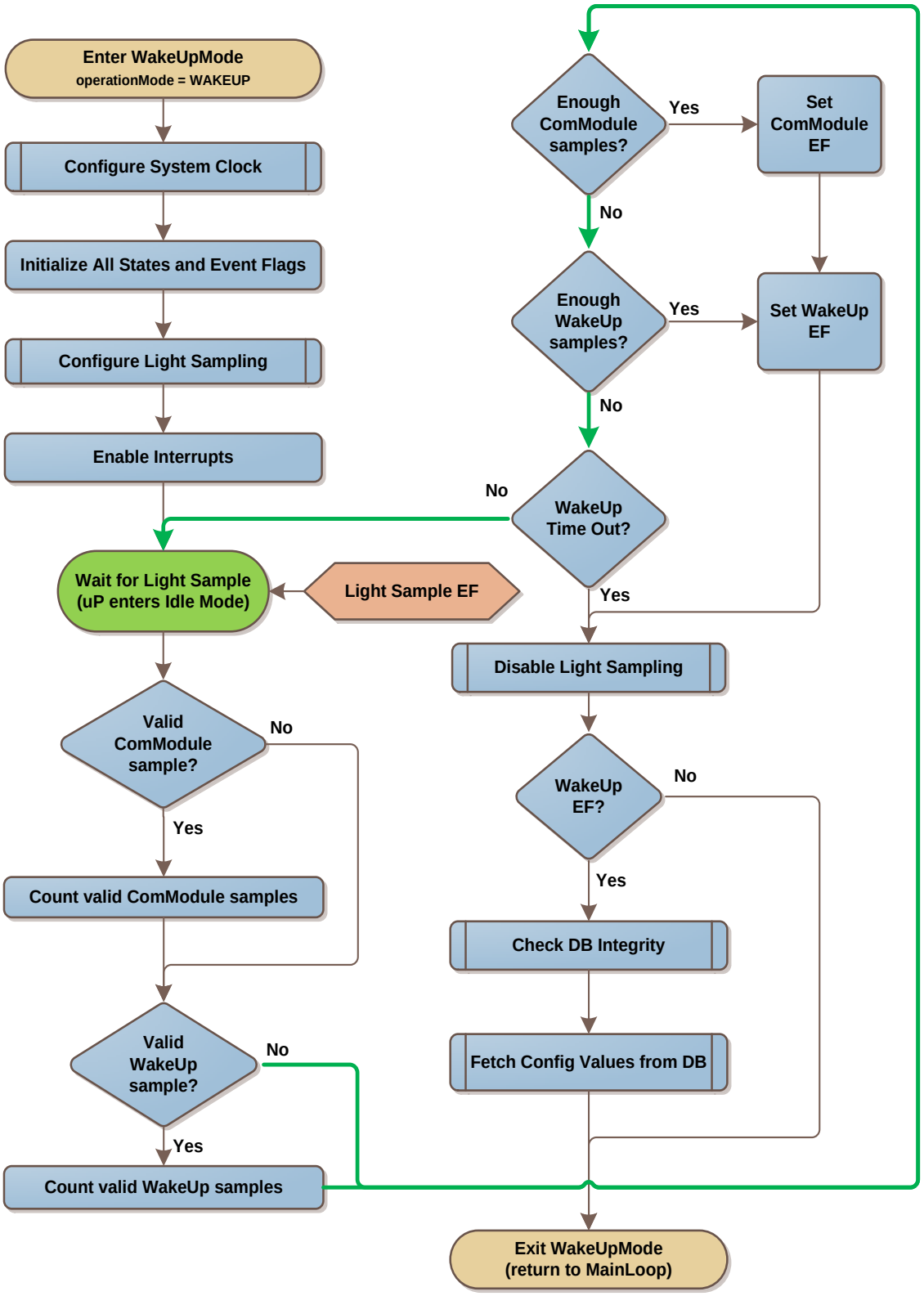

Figure 9: WakeUpMode Process

3.8 CommandMode Process

CommandMode (see Figure 10) processes all commands received from the *ComModule*. **MainLoop** handles detecting the presence of the *CM*, receiving and parsing the command string, and then handing off execution of the command to **CommandMode**. **CommandMode** executes the requested command.

There are four basic commands:

1) Read EEPROM Page
2) Read Ball Record
3) Write EEPROM Page
4) Set EEPROM Defaults

Command reception and processing is implemented within a loop structure in **MainLoop** so that multiple commands can be received and processed during a single *SenseModule* wake cycle. After all command processing has been completed, or after a command error has been detected, **CommandMode** terminates, and returns control to **MainLoop**.

Appendix C (page 118) details the communication protocols used between the *ComModule* and the *SenseModule*, including the *ComModule* detection protocol, and the implementation and timing diagrams for the software UART.

3.8.1 Read EEPROM Page Command

The Read EEPROM Page command reads 128-byte EEPROM pages starting at the requested page, and transmits them to the *CM*, one page at a time. After the *CM* receives the transmitted page, it can either terminate the command, or it can issue a "next page" response, and the *SM* will transmit the next contiguous page in EEPROM memory. Thus, it is possible for the *CM* to request any number of contiguous EEPROM pages within one command.

3.8.2 Read Ball Record Command

The Read Ball Record command is a special version of the Read EEPROM Page command that requests EEPROM pages by specifying a Ball Record number, or by requesting the newest ball record. There is also a provision in the command for requesting the next ball record (which is the oldest ball record that the *CM* has not yet uploaded. The Read Ball Record command looks up the Ball Page address for the specified Ball Record, and then issues requests via the Read EEPROM command to retrieve the necessary EEPROM pages, one-at-a-time, for transmission to the *CM*.

3.8.3 Write EEPROM Page Command

The Write EEPROM Page command writes data to the EEPROM at the specified page. This is the only command that can also specify individual bytes within a page. The command is used primarily to configure the *SM*'s system parameters on the Configuration page.

3.8.4 Set Defaults Command

This command is used to return the *SM* back to a default state. It clears the entire EEPROM, and then sets the Configuration page and the Ball Pointer page to default values.

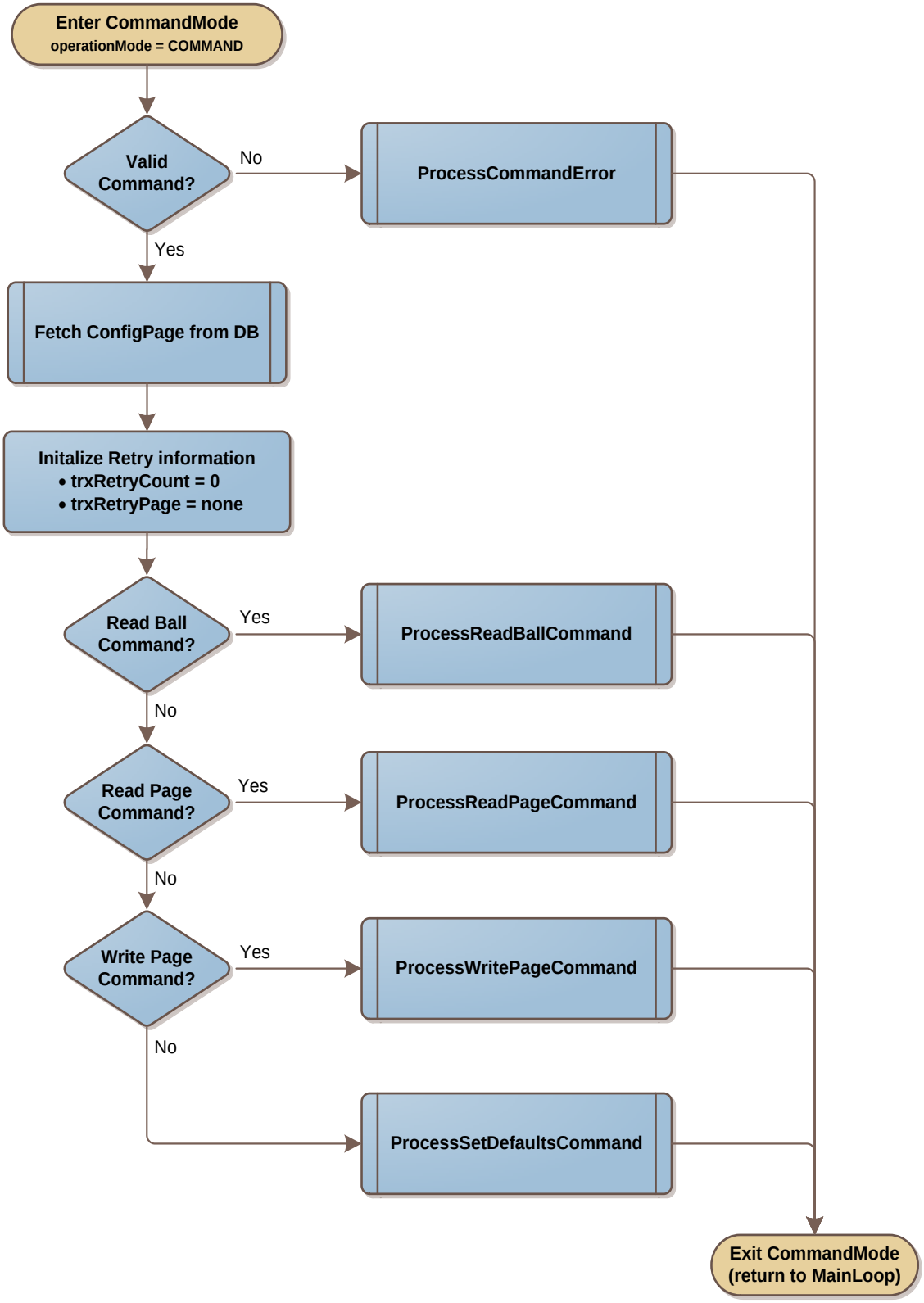

Figure 10: CommandMode Process

3.9 ApproachMode Process

ApproachMode (see Figure 11), followed by **SampleMode**, collaborate to detect, capture, and store the ambient light and 3-axis accelerometer sensor readings. **ApproachMode** manages the sampling process commencing with the bowler's approach, and continuing through release detection, at which time it hands the sampling duties off to **SampleMode**, which continues sampling during the ball's traversal of the lane, and through the ball's contact with the pins.

During **ApproachMode**, all samples are collected and stored within pages in the light and ADXL circular buffers. Those buffers accumulate the most recent three (3) seconds of sensor readings immediately preceding release of the ball, essentially implementing pre-trigger buffers, with release serving as the trigger. **ApproachMode** does not commit any data to EEPROM; rather it captures Light and ADXL samples and stores them in their respective circular buffers, whose contents it hands off to **SampleMode** after release is detected. **SampleMode** takes care of committing the sensor data collected during the bowler's approach to EEPROM while it continues sampling, placing the new samples at the heads of the circular buffers, while transferring sample pages from the tails of the buffers to EEPROM.

The *SenseModule* enters **ApproachMode** after **MainLoop** has determined that a valid wake-up condition occurred (***WakeUpEF*** is set), and that the presence of the *ComModule* was not detected within the *CM* detection period after waking up from **SleepMode**.

Before sampling commences, **ApproachMode** retrieves the location of the next available Ball Record Pointer and Ball Page from the Configuration Page in EEPROM. It then initializes the light and ADXL circular buffers, wakes up the light and ADXL sensors, configures the ADXL sensor, initializes the release and shutdown variables, and initiates waveform sampling.

ApproachMode then puts the μP into a low-power **IdleMode** while waiting for any of a number of sampling-related interrupts to occur. Those interrupts periodically cause the μP to vector from **IdleMode** to their respective interrupt vectors – it is possible to have additional interrupts trigger while processing the interrupt that pulled the μP out of **IdleMode**. Each triggered interrupt performs its function, and then sets one or more event flags, as appropriate.

Upon return from the last interrupt vector that was processed coming out of **IdleMode**, **ApproachMode** captures a copy of the current EFs, before resetting the EFs. Execution proceeds to the event flag processing loop, which processes all of the captured EFs that have been set since the last iteration of the EF loop (it is possible to have multiple EFs to process in a single loop). There is a fixed round-robin priority order to EF processing – processing of certain early (high-priority) EFs can trigger additional lower-priority EFs that will be processed later in the EF loop.

It is also possible for further interrupts to occur while processing the current set of EFs. Those additional interrupts, occurring subsequent to the beginning of EF processing, are processed as they occur, but any EFs they set are accumulated for the next iteration of the EF loop. Thus, when the current EF loop terminates, and **ApproachMode** is ready to return the μP to **IdleMode**, a check is first performed to

identify any additional EFs that may have accumulated during the previous iteration of the loop. If any such EFs exist, then the return to **IdleMode** is skipped, and a new iteration of the EF loop commences. Eventually, there will be no new EFs, and the μP will return to **IdleMode**.

The iterative process of waiting in **IdleMode**, and then processing interrupts and EFs continues until release is detected, or the release timeout expires. If release is not detected within the time out period, then all sampling processes are shutdown, the collected sample data is discarded, **ApproachMode** terminates, and execution returns to **MainLoop**, which skips **SampleMode**, and proceeds to **CleanUpMode** and ultimately back to **SleepMode**.

If release is detected, then ***ReleaseEF*** is set, sampling is allowed to continue, **ApproachMode** terminates, and execution returns to **MainLoop**, which then calls **SampleMode** to take over the sampling process. The EF loops for **ApproachMode** and **SampleMode** are quite similar, and both modes respond to the same interrupts. The main differences between the two processes are:

1) **ApproachMode** terminates upon detection of release, or upon reaching release timeout. **SampleMode** terminates upon detection of shutdown, or upon reaching shutdown timeout.
2) Both processes store samples at the heads of the sample buffers. However, **ApproachMode** does not transfer sample pages to EEPROM. Rather, its purpose is to implement a "pre-trigger" process that collects the samples immediately preceding release. As soon as **SampleMode** commences, it begins committing the contents of the circular buffers at the tails of the buffers to EEPROM, while continuing to store the latest sample data at the heads of the buffers. This is a classic producer-consumer problem: **SampleMode** must be able to transfer circular buffer pages to EEPROM (consume) faster than it stores new pages in those circular buffers (produce).

Processing of **ApproachMode** event flags proceeds in the following order:

3.9.1 *ProcessSampleClockEvent* (*SampleClockEF*)

SampleClockEF is set by the Light Sample Timer interrupt and serves as the "clock tick" for release timeout. ***ProcessSampleClockEvent*** counts the Sample Clock ticks, and checks the count against the release time out value. If the release time out expires before release is detected, ***ApproachModeTOEF*** is set, sampling is disabled, and **ApproachMode** terminates.

3.9.2 *ProcessLightSamplesEvent* (*LightSamplesEF*)

LightSamplesEF is set by the ADC0 interrupt (Light Sampling ADC) when the Light Samples buffer is full. ***ProcessLightSamplesEvent*** transfers the buffer contents to the Light Page at the head of the Light circular buffer. When the current Light page is full, it advances the head pointer to the next page. The Light release condition is checked here. If Light release is detected, ***LightReleaseEF*** is set.

3.9.3 ProcessADXLWatermarkEvent (*ADXLWatermarkEF*)

ADXLWatermarkEF is set by the Port 0 Interrupt when the ADXL345 issues a Watermark interrupt upon reaching 25 ADXL samples in its internal buffer. If ***ProcessSamplePageEvent*** has not already initiated

the ADXL page transfer, then ***ADXLReadPageEF*** is set for processing by ***ProcessI2CControlEvent*** later in the EF loop.

3.9.4 *ProcessADXLSampleEvent* (*ADXLSampleEF*)

ADXLSampleEF is set by the SMBUS0 interrupt (I^2C) whenever it has buffered and compressed an ADXL sample into 5-bytes. ***ProcessADXLSampleEvent*** transfers the 5-byte sample to the current ADXL sample page. If the current sample page has been filled, this routine advances the ADXL circular buffer head pointer to the next page. The ADXL release condition is tracked here, and if ADXL release is detected, ***ADXLReleaseEF*** is set.

3.9.5 *ProcessI2CControlEvent* (all EFs)

ProcessI2CControlEvent is the "traffic cop" for the I^2C bus (SMBUS0). The I^2C bus is a shared resource between the ADXL345 and the serial EEPROM, and connects both the ADXL345 and the EEPROM to the microprocessor. ***ProcessI2CControlEvent*** is in charge of prioritizing competing requests, and assigns and manages "ownership" of the I^2C bus. The routine also helps manage the Light and ADXL Circular Page Buffers. After this routine assigns "ownership" of the I^2C bus, ***ProcessSamplePageEvent*** handles initiating the actual transfer of individual sample pages over the I^2C bus.

There are three types of actions competing for the I^2C resource: ADXL Page Reads, Light Page Writes, and ADXL Page Writes. However, since there are no Light or ADXL Page Writes to EEPROM during **ApproachMode**, only ADXL Page Reads can occupy the I^2C bus during **ApproachMode**.

3.9.6 *ProcessSamplePageEvent* (I^2C mutex, I^2C retry)

ProcessSamplePageEvent takes care of initiating the transfer of data via the I^2C bus, based on the "ownership" of the I^2C bus as determined and assigned by ***ProcessI2CControlEvent***. If the I^2C mutex is available, or the last attempt to initiate I^2C communications failed (because the EEPROM was still busy writing the previous page), then ***ProcessSamplePageEvent*** calls the event processing routine for the I^2C action that currently has ownership of the I^2C bus. Since there are no Light or ADXL Page Writes to EEPROM during **ApproachMode**, ***ProcessSamplePageEvent*** only calls ***ProcessReadADXLPageEvent*** during **ApproachMode**.

3.9.7 *ProcessReadADXLPageEvent* (*ADXLReadPageEF*)

ProcessReadADXLPageEvent is called from ***ProcessSamplePageEvent***. It sets up the new ADXL page at the head of the ADXL circular buffer, and then initiates the I^2C bus request.

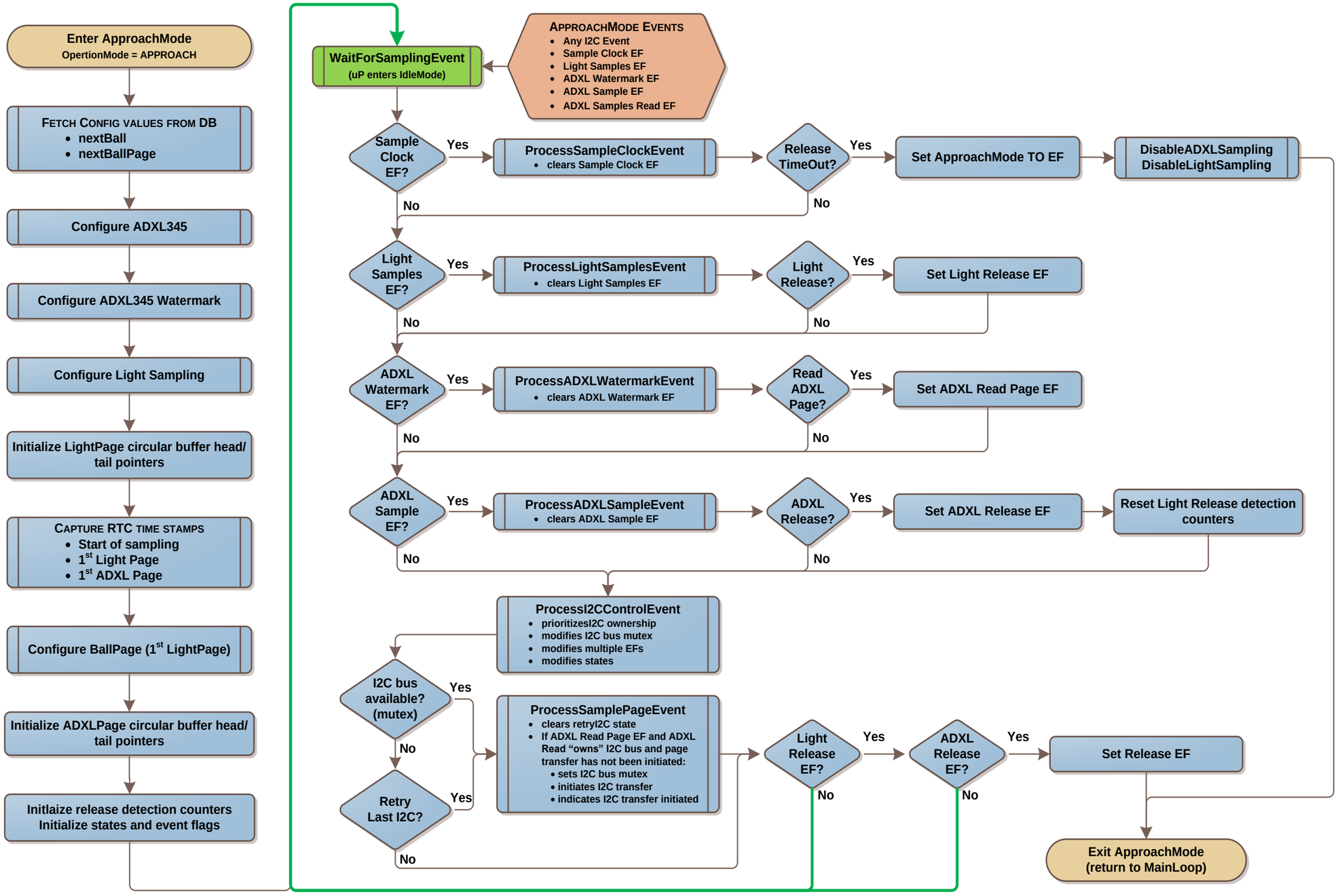

Figure 11: ApproachMode Process

3.10 SampleMode Process

SampleMode (see Figure 12) takes over where **ApproachMode** left off, essentially "inheriting" the state of the *SenseModule* from **ApproachMode**. Where **ApproachMode** managed collecting sample data during approach and release, **SampleMode** handles sampling from immediately after release through the ball's impact with the pins.

ApproachMode "primed" the Light and ADXL circular buffers with the most recent three (3) seconds of sensor readings that immediately preceded detection of release. After **ApproachMode** has detected a valid release condition, **SampleMode** takes care of committing (writing) the sensor data collected during **ApproachMode** to EEPROM while it also continues sampling, placing the new samples at the heads of the circular buffers, while transferring sample pages from the tails of the buffers to EEPROM.

The *SM* enters **SampleMode** after **MainLoop** has determined that a valid release condition occurred (***ReleaseDetectedEF*** is set) during **ApproachMode**. At the start of **SampleMode**, the sample page counters, sample time out detection, and sample shutdown detection are all initialized.

As with **ApproachMode**, **SampleMode** returns the μP to **IdleMode** while waiting for any of a number of sampling-related interrupts to occur. Each triggered interrupt performs its function, and then sets one or more event flags, as appropriate. Upon return from the last interrupt vector that was processed coming out of **IdleMode**, **SampleMode** captures a copy of the current EFs, before resetting the EFs.

Execution proceeds to the **SampleMode** event flag processing loop, which processes all of the captured EFs that have been set since the last iteration of the EF loop (it is possible to have multiple EFs to process in a single loop). There is a fixed round-robin priority order to EF processing – processing of certain early (high-priority) EFs can trigger additional lower-priority EFs that will be processed later in the EF loop.

It is also possible for further interrupts to occur while processing the current set of EFs. Those additional interrupts, occurring subsequent to the beginning of EF processing, are processed as they occur, but any EFs they set are accumulated for the next iteration of the EF loop. Thus, when the current EF loop terminates, and **SampleMode** is ready to return the μP to **IdleMode**, a check is first performed to identify any additional EFs that may have accumulated during the previous iteration of the loop. If any such EFs exist, then the return to **IdleMode** is skipped, and a new iteration of the **SampleMode** EF loop commences. Eventually, there will be no new EFs, and the μP will return to **IdleMode**.

The iterative process of waiting in **IdleMode**, and then processing interrupts and EFs continues until automatic shutdown is detected (***LightShutdownEF*** and ***ADXLShutdownEF*** are set), or the sample timeout expires before shutdown is detected (***SampleModeTOEF*** is set). In either case, **SampleMode** continues writing sample pages to EEPROM until the circular buffers are both empty. All sampling processes are shutdown, and after the conclusion of sample page transfer to EEPROM, ***NewBallEF*** is set to indicate that there is New Ball Record stored in EEPROM. Execution then returns to **MainLoop**, where **CleanUpMode** takes care of updating the Configuration Page, before the *SM* returns to **SleepMode**.

The event flag loop for **SampleMode** is quite similar to that for **ApproachMode**, and responds to the same interrupts. However, **SampleMode** handles the following additional events:

1) **SampleMode** both collects new sample pages, as well as commits (writes) previously collected sample pages to EEPROM. Thus, it must manage three tasks that utilize the I^2C bus resource. As soon as **SampleMode** commences, it begins committing the current page contents of the circular buffers to EEPROM, beginning at the tails of the buffers, while continuing to store the latest sample data to the pages at the heads of the buffers. This is a classic producer-consumer problem: **SampleMode** must be able to consume pages (transfer circular buffer pages to EEPROM), faster than it produces pages (stores new pages in those circular buffers).
2) **SampleMode** detects the automatic sampling shutdown condition, as well as tracks sampling timeout.

Processing of **SampleMode** event flags proceeds in the following order:

3.10.1 *ProcessSampleClockEvent* (*SampleClockEF*)

SampleClockEF is set by the Light Sample Timer interrupt and serves as the "clock tick" for automatic sampling shutdown detection. The Light and ADXL time out events are captured elsewhere, by checking for the maximum number of pages that can be captured during sampling.

3.10.2 *ProcessLightSamplesEvent* (*LightSamplesEF*)

LightSamplesEF is set by the ADC0 interrupt (Light Sampling ADC) when the Light Samples buffer is full. ***ProcessLightSamplesEvent*** transfers the buffer contents to the Light Page at the head of the Light circular buffer. When the current Light page is full, it advances the head pointer to the next page. The Light shutdown condition is checked here. If Light shutdown is detected, ***LightShutdownEF*** is set.

3.10.3 ProcessADXLWatermarkEvent (*ADXLWatermarkEF*)

ADXLWatermarkEF is set by the Port 0 Interrupt when the ADXL345 issues a Watermark interrupt upon reaching 25 ADXL samples in its internal buffer. If neither the ADXL sampling shutdown condition nor the ADXL sampling timeout condition have been detected, then ***ADXLReadPageEF*** is set for processing by ***ProcessI2CControlEvent*** later in the EF loop.

3.10.4 *ProcessADXLSampleEvent* (*ADXLSampleEF*)

ADXLSampleEF is set by the SMBUS0 interrupt (I^2C) whenever it has buffered and compressed an ADXL sample into 5-bytes. ***ProcessADXLSampleEvent*** transfers the 5-byte sample to the current ADXL sample page. If the current sample page has been filled, this routine advances the ADXL circular buffer head pointer to the next page. The ADXL sampling shutdown condition is tracked here, and if detected, ***ADXLReleaseEF*** is set, and ADXL sampling is disabled.

3.10.5 *ProcessI2CControlEvent* (all EFs)

ProcessI2CControlEvent is the "traffic cop" for the I^2C bus (SMBUS0). The I^2C bus is a shared resource between the ADXL345 and the serial EEPROM, and connects both the ADXL345 and the EEPROM to the microprocessor. ***ProcessI2CControlEvent*** is in charge of prioritizing competing requests, and assigns and manages "ownership" of the I^2C bus. The routine also helps manage the Light and ADXL Circular Page

Buffers, and the Light and ADXL sampling timeouts. After this routine assigns ownership of the I^2C bus, ***ProcessSamplePageEvent*** handles initiating the actual transfer of individual sample pages over the I^2C bus.

There are three types of events competing for the I^2C resource: ADXL Page Reads (***ADXLReadPageEF***), Light Page Writes (***LightWritePageEF***), and ADXL Page Writes (***ADXLWritePageEF***). ***ProcessI2CControlEvent*** processes and prioritizes all three. When it detects that the I^2C bus is available for assignment (has completed its most recent task, or is idle when an EF comes in), it assigns new ownership based on the following prioritization conditions:

1) ***ADXLReadPageEF*** has the highest priority, as the ADXL345's internal sample buffer must be read before it overflows. The ***ADXLWatermarkEF*** is issued when the ADXL sample buffer has reached 25 samples (every 125 ms). The ADXL sample buffer can hold a maximum of 32 samples, which leaves 35 ms to start reading the buffer contents (at 200 Hz, 5 ms per sample).
2) ***LightWritePageEF*** receives a higher priority than ***ADXLWritePageEF*** under one condition. The first Light page also doubles as the Ball Page (the first page of any Ball Record). Thus, the first Light page must be the first page written to EEPROM for a new Ball Record.
3) ***ADXLWritePageEF*** and ***LightWritePageEF*** have equal priorities, with the exception of 2) above. When both types of pages are available in the sample buffers, priority alternates between the ***ADXLWritePageEF*** and the ***LightWritePageEF***, so that neither event suffers from "starvation", which prevents overflow of either circular buffer, while the other is being serviced. In actuality, there are eight (8) ADXL pages generated for each Light page, since an ADXL page fills up in 125 ms, as opposed to a Light page, which takes 1 second to fill up. However the alternate page writing scheme is sufficient to prevent starvation.

3.10.6 *ProcessSamplePageEvent* (I^2C mutex, I^2C retry)

ProcessSamplePageEvent takes care of initiating the transfer of data via the I^2C bus, based on the ownership of the bus as determined and assigned by ***ProcessI2CControlEvent***. If the I^2C mutex is available, or the last attempt to initiate I^2C communications failed (because the EEPROM was still busy writing the previous page), then ***ProcessSamplePageEvent*** calls the event processing routine for the I^2C action that currently has ownership of the I^2C bus. Those event processing routines are ***ProcessReadADXLPageEvent***, ***ProcessWriteLightPageEvent***, and ***ProcessWriteADXLPageEvent***.

3.10.7 *ProcessReadADXLPageEvent* (*ADXLReadPageEF*)

ProcessReadADXLPageEvent is called from ***ProcessSamplePageEvent***. If ADXL sampling has not timed out, it sets up the new ADXL page at the head of the ADXL circular buffer, and then initiates the I^2C bus request.

3.10.8 *ProcessWriteLightPageEvent* (*LightWritePageEF*)

ProcessWriteLightPageEvent is called from ***ProcessSamplePageEvent***. It stores the current Ball Record page number into the Light Page located at the tail of the buffer, sets the Light Buffer page location to be written to EEPROM, as well as the destination EEPROM page address, and initiates the I^2C request.

3.10.9 *ProcessWriteADXLPageEvent* (*ADXLWritePageEF*)

ProcessWriteADXLPageEvent is called from ***ProcessSamplePageEvent***. It stores the current Ball Record page number into the ADXL Page located at the tail of the buffer, sets the ADXL Buffer page location to be written to EEPROM, as well as the destination EEPROM page address, and initiates the I^2C request.

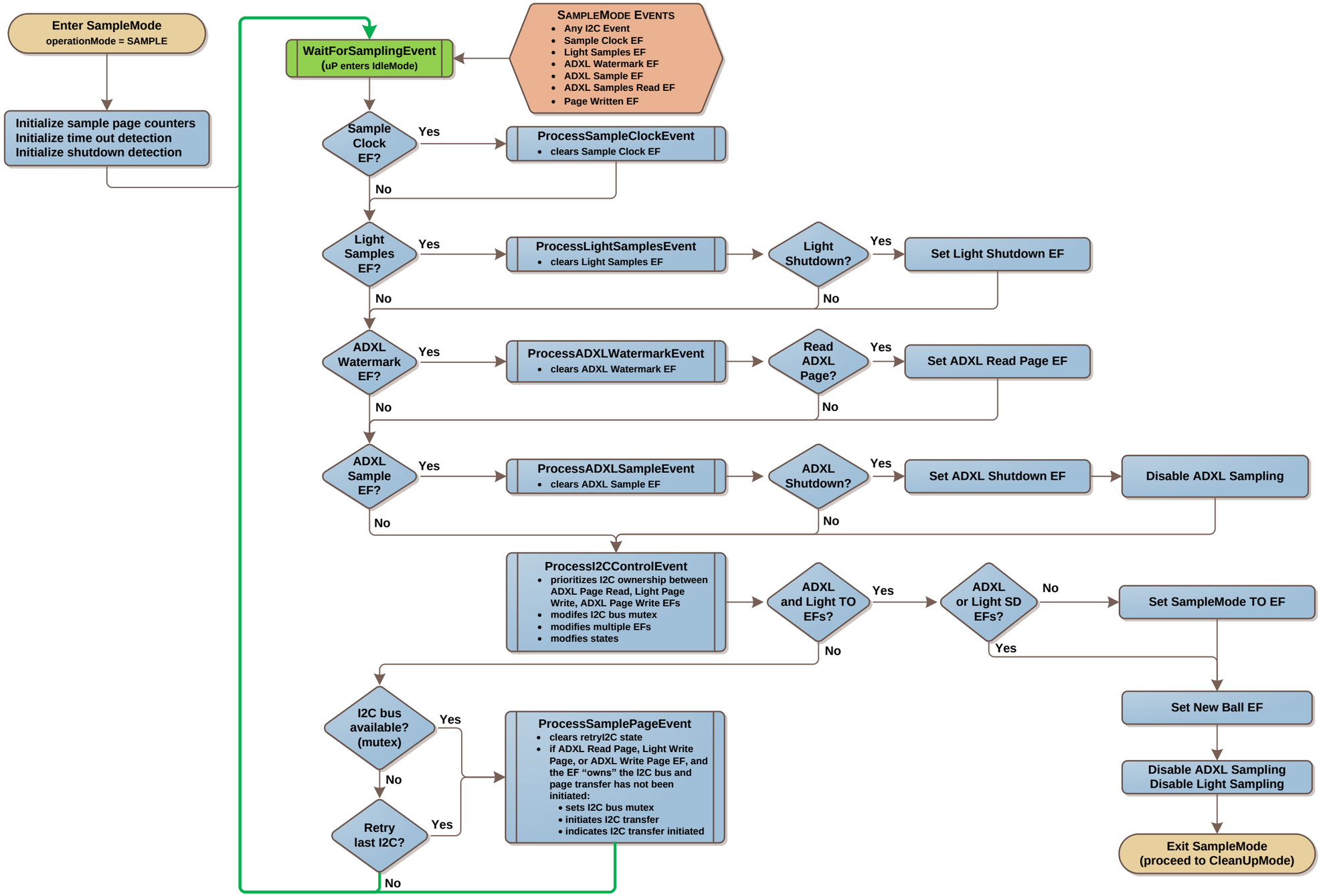

Figure 12: SampleMode Process

3.11 CleanUpMode Process

MainLoop calls **CleanUpMode** (see Figure 13) to handle any remaining processing after coming out of **WakeUpMode**, **CommandMode**, **ApproachMode**, or **SampleMode**.

Currently, there is nothing additional to be done when coming into **CleanUpMode** from **WakeUpMode** or **ApproachMode**, and **CleanUpMode** returns immediately to **MainLoop** so that the *SenseModule* can return to **SleepMode**.

3.11.1 CommandMode Clean Up

Following the termination of **CommandMode**, **CleanUpMode** turns off serial reception and transmission, and retrieves the Ball Pointer Page from EEPROM, and generates a new count of the new and deleted Ball Records. New ball records are those that have not yet been uploaded to the *ComModule*. Deleted Ball Records are those records that have been at least partially overwritten by more recent Ball Records.

CleanUpMode then retrieves the Configuration Page from EEPROM so that it can update the following Ball Record database parameters:

- *firstNewBall*
- *newBallCount*
- *deletedBallCount*

After the above parameters have been updated, the Configuration Page is written back to EEPROM, and **CleanUpMode** terminates and returns to **MainLoop**, so that the *SenseModule* can return to **SleepMode**.

3.11.2 SampleMode Clean Up

Following the termination of **SampleMode**, **CleanUpMode** first checks that a new ball has been captured. As *SenseModule* development continues, there will be some post-processing of the potential new ball record to verify that it resulted from a legitimate waveform, and not from a false start-up condition, such as that generated during the return of the ball from the pin setting machine, through the subway, to the ball return. Such false start-up conditions have already been observed on occasion during initial *SM* testing.

Currently, whenever **SampleMode** terminates, it indicates that it has captured a new ball, and **CleanUpMode** retrieves the Ball Page for the new ball from EEPROM (first page of the new Ball Record), and updates the following Ball Record information for the new ball:

- Ball Page header
- *sampleCount*
- *ballDate*
- *ballTimeStamp*
- *endTimeStamp*
- *lightPages*
- *adxlPages*

After the Ball Page has been updated, **CleanUpMode** commits it back to EEPROM.

CleanUpMode then retrieves the Ball Pointer Page from EEPROM, and updates all of the Ball Record Pointers for which any of their corresponding Ball Record pages were overwritten/deleted by the new Ball Record. The Ball Pointer entry for the new ball is then updated, and the Ball Pointer Page is committed back to EEPROM.

The last **CleanUpMode** task is to retrieve the Configuration Page from EEPROM and update the following database parameters:

- *ballCount*
- *newestBall*
- *nextBallPage*
- *fristNewBall*
- *newBallCount*
- *deletedeBallCount*
- *nextBall*

After the above parameters have been updated, the Configuration Page is written back to EEPROM, and **CleanUpMode** terminates, so that the *SenseModule* can return to **SleepMode**.

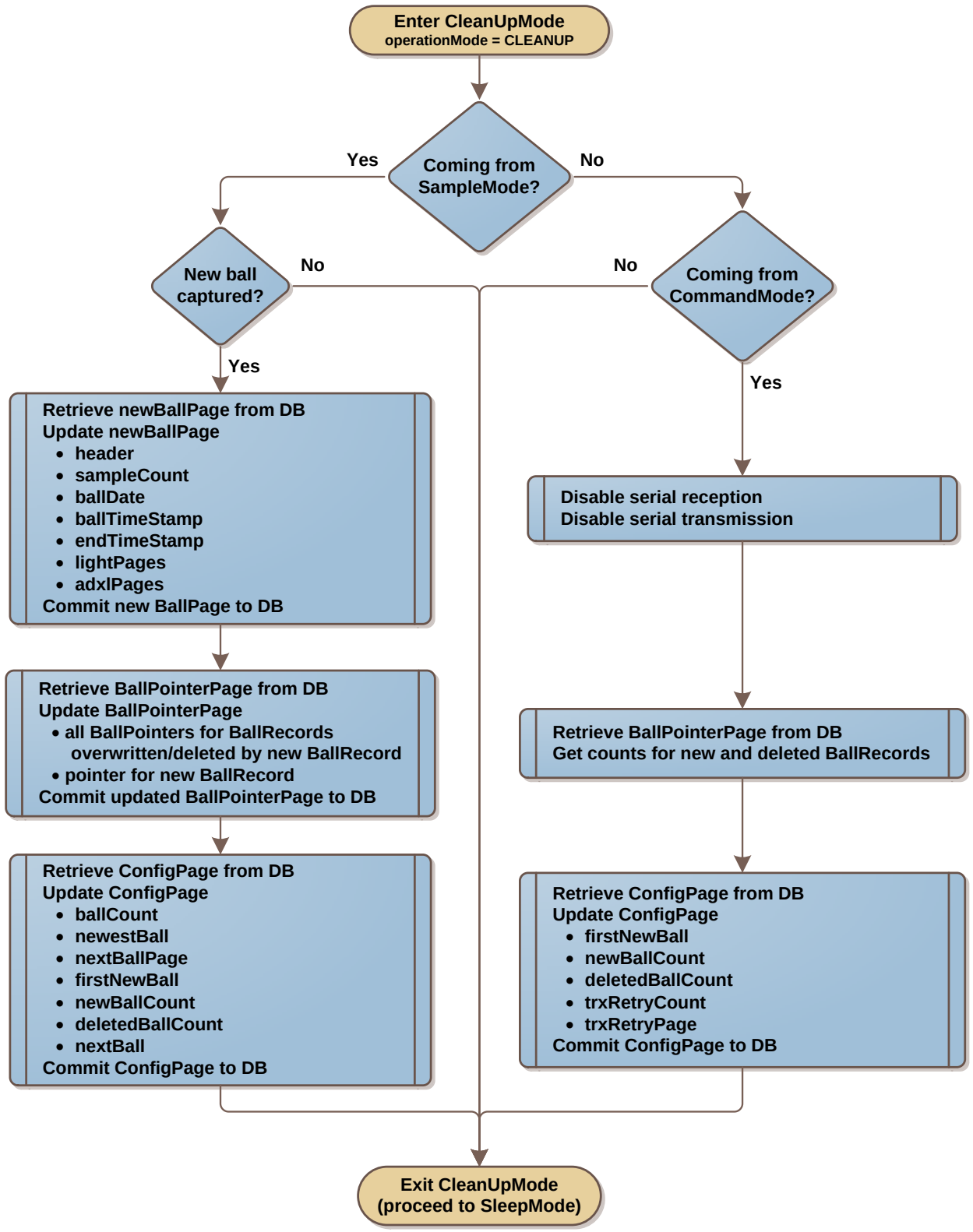

Figure 13: CleanUpMode Process

3.12 Sampling Data Flow

The *SenseModule*'s main function is as a data collection platform for its two sensors:

1) Ambient Light Sensor (TSL13 Light-to-Voltage converter)
2) Acceleration and Tilt Sensor (ADXL345 3-axis accelerometer)

The sampling intervals and reading of the sensor data are all interrupt-driven, while storage of that sensor data is event-driven. The following section describes how the various interrupts, events, processes, and data structures relate to each other.

3.12.1 Ambient Light Sampling and Storage

The ambient light sampling data flow diagram is given in Figure 14. All ambient light samples are initiated by an interrupt from the 240 Hz Light Sample Timer, which has two phases. The first phase supplies power to the TSL13 Light-to-Voltage converter and sets the timer to interrupt again 100 μs later, allowing the TSL13's output to stabilize after power-up.

The second phase sets the timer back to phase 1 (240 Hz), and starts the ADC0 conversion process that samples the TSL13 output. When the ADC conversion is complete, ADC0 issues a conversion complete interrupt. The ADC0 interrupt starts the data flow process for light samples.

The ADC0 interrupt has three phases. The first two phases alternate every 4.167 μs (240 Hz). The first phase captures the first 240 Hz light sample of a pair of samples to be summed together. The second phase captures the second sample of that pair, adds it to the first sample, takes the average of the two samples, and stores the averaged reading into a 12-element sample buffer for later transfer to the Light Page buffer. Thus, the resulting 120 Hz samples have been low-pass filtered to remove the 120 Hz ripple induced in the ambient light readings by the overhead fluorescent lighting installed in most bowling establishments. The second phase issues ***SampleClockEF***.

The third phase occurs coincident with the second phase when the ADC0 Sample Buffer is full (every 100 ms, 12 averaged samples). The ADC0 interrupt transfers the buffer contents to the Light Sample Buffer and issues ***LightSampleEF***.

ProcessSampleClockEvent is called from within the **ApproachMode** and **SampleMode** event processing loops when ***SampleClockEF*** is set, which counts the time stamps, and also takes care of tracking the **ApproachMode** time out function while in **ApproachMode**.

ProcessLightSamplesEvent is called from within the **ApproachMode** and **SampleMode** event processing loops when ***LightSamplesEF*** is set, and takes care of transferring the 12 Light samples from the Light Sample Buffer to the Light Page at the head of the Light Page circular buffer. During **SampleMode**, when the current Light Page has filled up (1 sec), ***LightPageEF*** is set, so that ***ProcessI2CControlEvent*** can process the new Light Page.

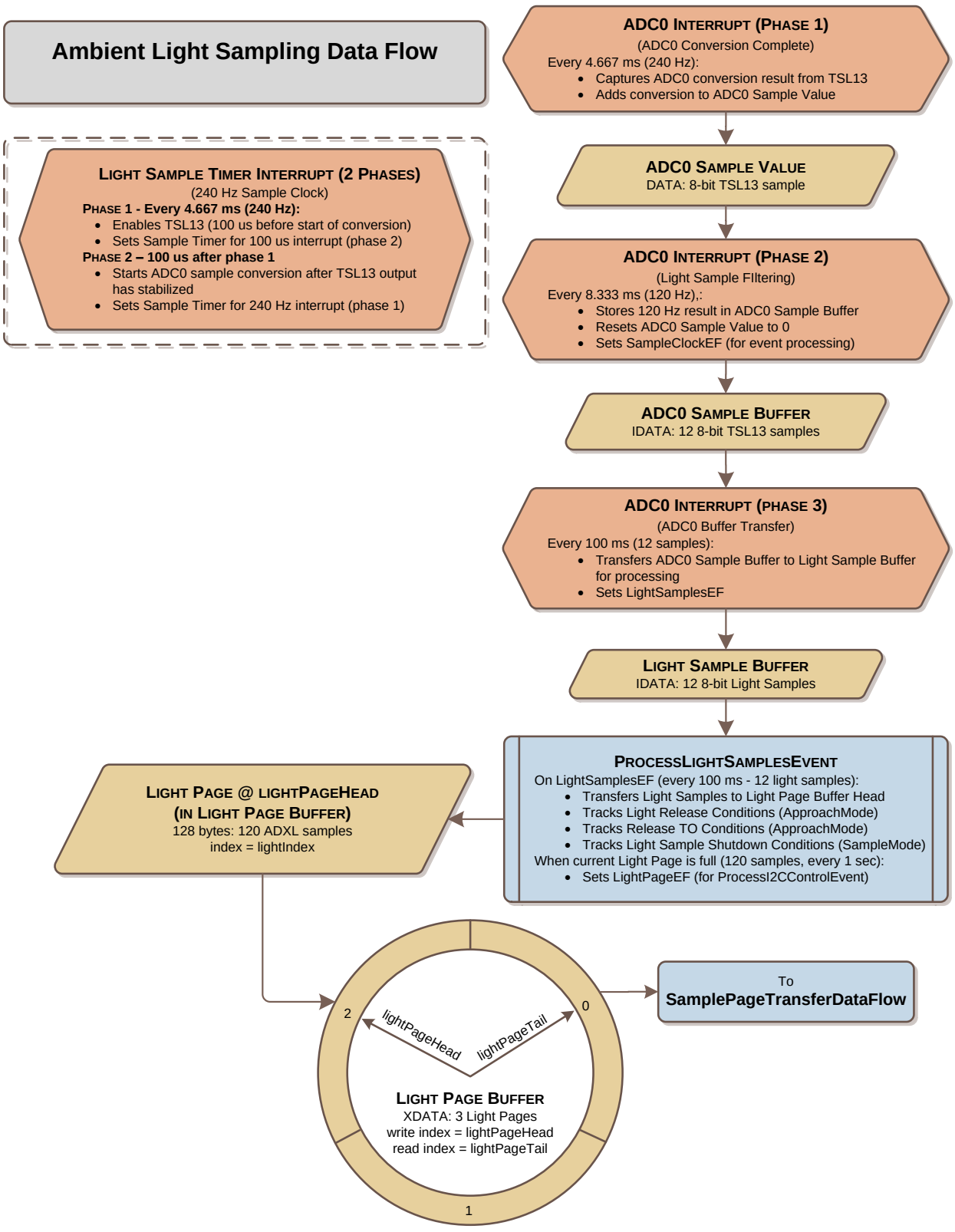

Figure 14: Light Sampling Data Flow Diagram

3.12.2 Acceleration Sampling and Storage

The acceleration sampling data flow diagram is given in Figure 15. The ADXL345 collects 3-axis acceleration samples autonomously at ~200 Hz, and accumulates the samples in its internal sample buffer. When that buffer reaches 25 samples, the ADXL345 issues an interrupt via an external pin to the µP, which the Port 0 PortMatch interrupt interprets as an ADXL Watermark event. That interrupt sets ***ADXLWatermarkEF***, saves the last captured RTC time as the start time for the ADXL345's current buffer contents, and then captures the current RTC value for the next ADXL Watermark interrupt.

ProcessWatermarkEvent is called from within the **ApproachMode** and **SampleMode** event processing loops when ***ADXLWatermarkEF*** is set, and if ADXL sampling has not already been shut down or timed out, then it sets ***ADXLReadPageEF***. While in **SampleMode**, it also counts the number of ADXL Watermark events as an approximation for the total time spent in **SampleMode** (125 ms precision).

ProcessI2CControlEvent is called from within the **ApproachMode** and **SampleMode** event processing loops, and initiates transfer of the ADXL345 sample buffer contents via SMBUS0 when the I^2C bus next becomes available.

The SMBUS0 interrupt handles retrieving the ADXL345 sample buffer contents using two phases. Phase 1 of the SMBUS0 interrupt reads sample bytes, one at a time, from the ADXL345's buffer, and accumulates them in the interrupt's ADXL Buffer, until that buffer contains a complete ADXL sample (6 bytes). Phase 2 of the SMBUS0 interrupt transfers those bytes to the ADXL Sample Buffer, and sets the ***ADXLSampleEF***, so that ***ProcessADXLSampleEvent*** can then continue processing the ADXL sample during event processing.

ProcessADXLSampleEvent is called from within the **ApproachMode** and **SampleMode** event processing loops when ***ADXLSampleEF*** is set, and compresses the 6-byte ADXL sample in place into a 5-byte sample. It then stores the sample in the ADXL Page at the head of the ADXL Page circular buffer. ***ProcessADXLSampleEvent*** also tracks the **ApproachMode** ADXL Release condition, and the **SampleMode** ADXL Shutdown condition, and initiates the transfer of the next ADXL sample, if the current ADXL page has not been filled. When the page has been filled, ***ADXLPageEF*** is issued so that ***ProcessI2CControlEvent*** can process the new ADXL page.

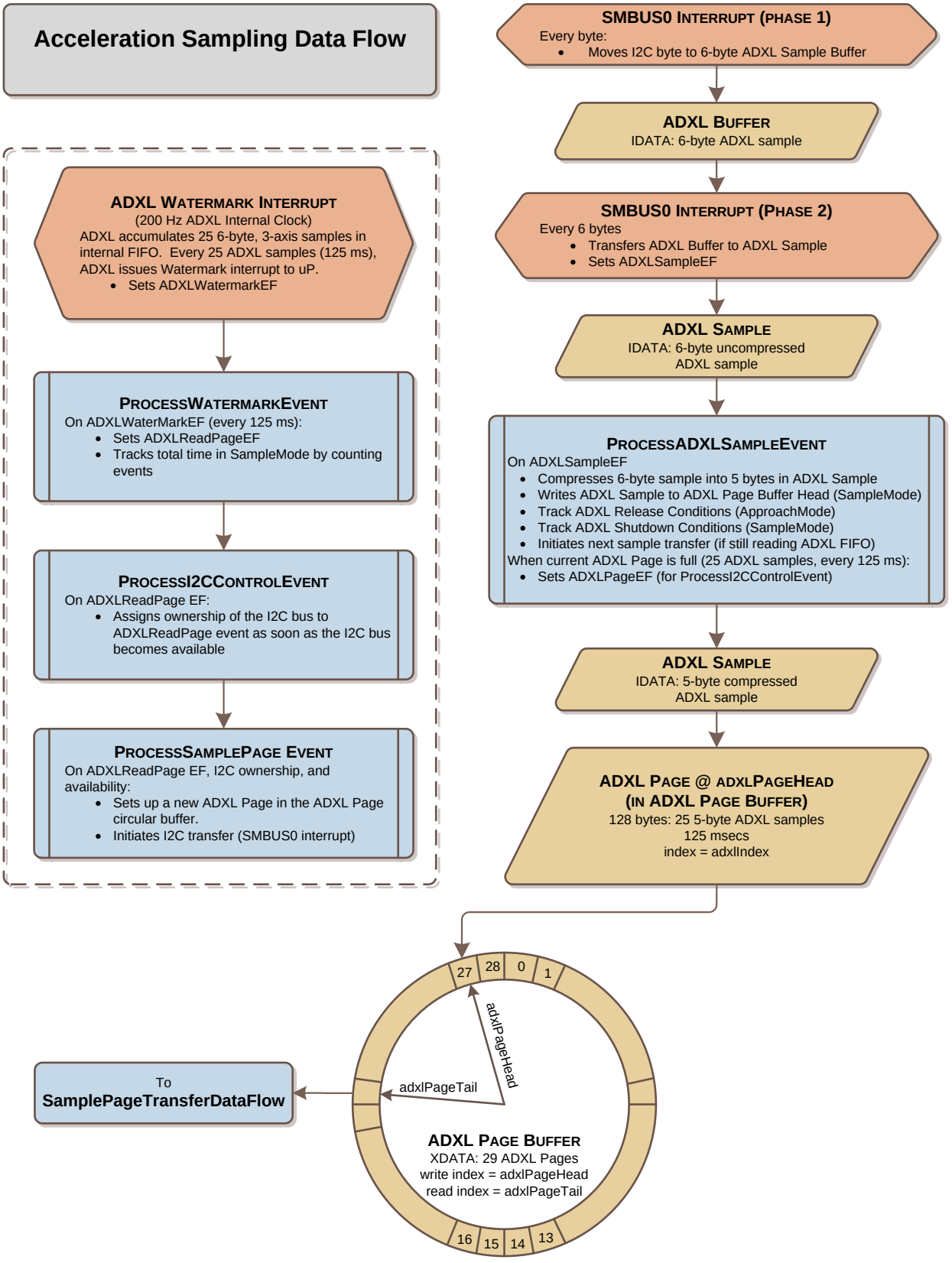

Figure 15: Acceleration Sampling Data Flow Diagram

3.12.3 Sample Page Transfer and Storage

The sample page transfer data flow diagram is given in Figure 16. ***ProcessI2CControlEvent*** (3.10.5) is called on every iteration of the **SampleMode** event processing loop, after the primary data collection events have been processed. This routine prioritizes competing I^2C requests and assigns and manages "ownership" of the I^2C bus. It finds the next available pages in the Light and ADXL circular buffers, and hands the locations of those pages off to ***ProcessSamplePageEvent*** (3.10.6) to initiate the I^2C transfers.

During **ApproachMode**, there is only one active source for I^2C events: ADXL Read Page events (***ADXLReadPageEF***). Thus, there is no competition for the I^2C bus during **ApproachMode**.

During **SampleMode**, however, there are three sources competing for the I^2C bus:

1) ADXL Read Page events (***ADXLReadPageEF***)
2) Light Write Page events (***LightWritePageEF***)
3) ADXL Write Page events (***ADXLWritePageEF***)

ADXL Read Page events are initially triggered by ADXL Watermark events, and the page is transferred to the μP from the EEPROM via the I^2C bus as a 150-byte stream (25 6-byte samples) – 150 separate SMBUS0 interrupts must be processed in order to retrieve the ADXL345 buffer contents.

Light Write Page events are triggered whenever the Light Page circular buffer is not empty. Since that buffer is full coming out of **ApproachMode**, Light Write Page events start being issued immediately after **SampleMode** starts.

ADXL Write Page events are triggered whenever the ADXL Page circular buffer is not empty. Since that buffer is also full coming out of **ApproachMode**, ADXL Write Page events also start being issued immediately after **SampleMode** starts.

Thus, the start of **SampleMode** is the busiest time for the I^2C bus, as **SampleMode** is constantly striving to catch up on writing Light and ADXL pages from the tails of their respective circular buffers to the new Ball Record in EEPROM, while also transferring new ADXL pages from the ADXL345's sample buffer to the head of the ADXL Page circular buffer, and storing new Light Samples at the head of the Light Page circular buffer.

The Ball Record array located in EEPROM is also one large circular buffer. It spans 1022 pages, from EEPROM page 2 to EEPROM page 1023. Ball Records are variable length, consisting of a Ball Page, and a mix of Light Pages and ADXL Pages. The contents of those pages all come from the Light Page and ADXL Page circular buffers, transferred via the I^2C bus. The pages of a new Ball Record are written contiguously to EEPROM, overwriting the existing contents. When EEPROM page 1023 is written, the Configuration Page (page 0), and the Ball Pointer Page (page 1) are skipped over, and the next available sample page is page 2.

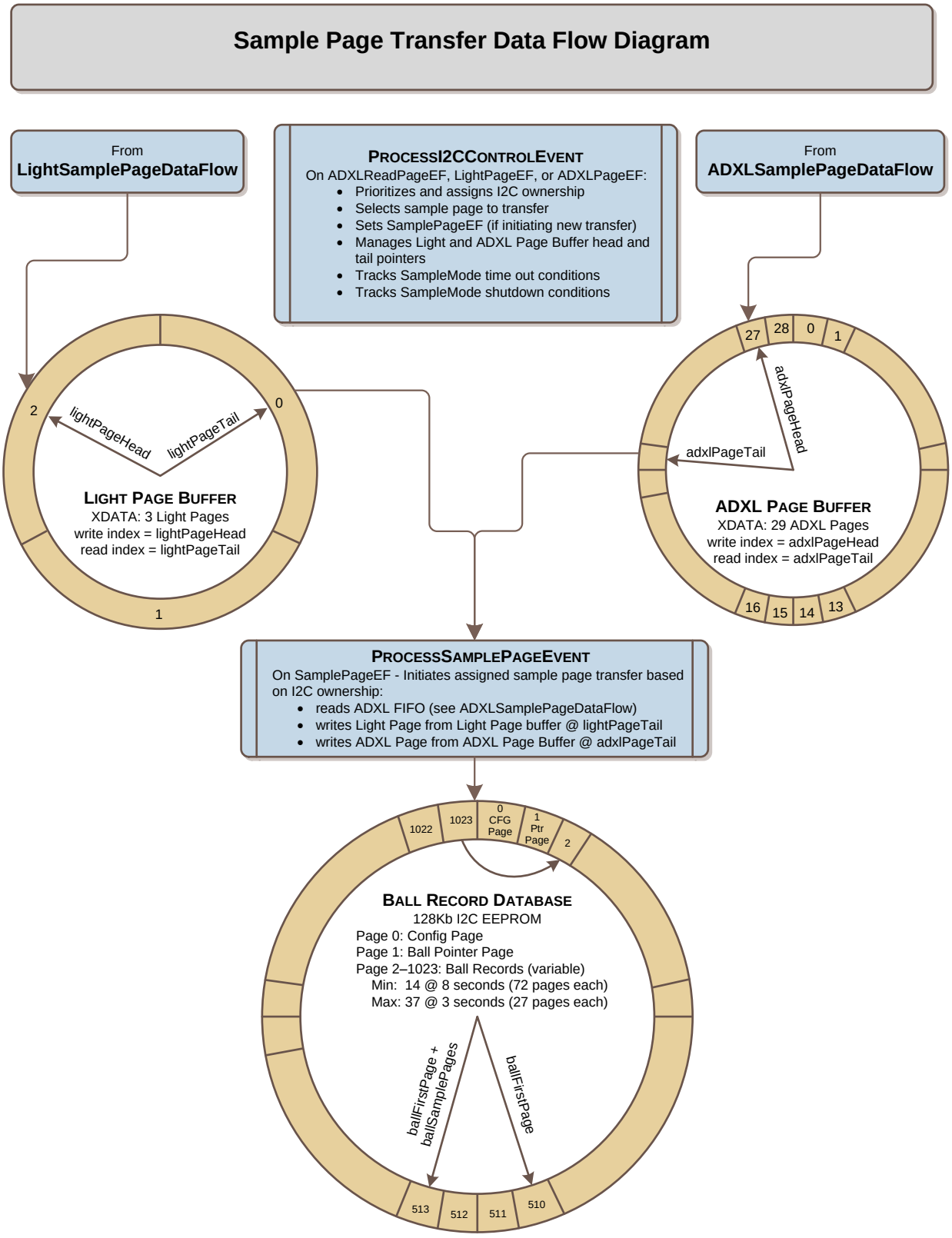

Figure 16: Sample Page Transfer Data Flow Diagram

Section IV: *SenseModule* Performance and Raw Data Collection

Although the *SenseModule* collects ambient light data similar to that of the original *SMARTDOT* module, the 3-axis accelerometer waveforms that the *SM* was designed to record had not yet been collected or observed. Additionally, the morphology, phase, and interrelation of those waveforms were needed in order to develop the detection algorithms required for automatic operation of the *SM*. Thus, the *SenseModule* has evolved through several iterations of hardware and embedded software from bench-top prototype, to its initial incarnation as a manual raw data collection platform, to the fully functional device presented in this paper. Those three phases of development are presented below.

1) **Bench-Top Prototype:** The initial development of the *SenseModule* and the *ComModule* was conducted on breadboard prototypes utilizing F930DK development kits from Silicon Laboratories [23]. The embedded software for both the *SM* and the *CM* was developed using the Silicon Labs IDE, along with Keil's assembler and C51 compiler [18]. Using the breadboard prototypes, it was possible to evaluate the feasibility of the hardware design, develop the fundamental architecture of the embedded software, the EEPROM data structures, and work out the communication scheme between the *SM*, the *CM*, and the PC.
2) **First-Look Data Collection:** Until the first *SenseModule* made its way down the lane in a bowling ball, only the very basic morphology of the 3-axis acceleration waveforms could be accurately predicted. Ultimately, real-world acceleration data had to be collected to gain a better understanding of those waveforms before a truly autonomous *SM* could be developed. The first *SM* prototypes that could be placed under a finger insert in the ball were developed based on the hardware design and embedded software from the bench top breadboard. The schematic and printed circuit board drawings were generated using Eagle PCB layout software [19]. In April of 2010, Advanced Circuits [30] manufactured the *SenseModule* printed circuit boards from Gerber files generated from the schematic shown in Figure 2, and then Advanced Assembly [31] assembled five *SM* prototypes using those PCBs. The earliest *SM* versions operated only in a manual, one-shot mode, and were capable of collecting and storing just a single set of ambient light and acceleration waveforms at a time. Several rounds of real-world data collection were conducted from early-May through mid-June of 2010.
3) **Autonomous Operation:** The raw data waveforms collected during the rounds of testing in (2) were then used to devise the first automated routines for the *SenseModule*. The fully automated functionality of the *SM* evolved through iterative development and testing from late-June through late-August of 2010, resulting in the *SM*'s automatic start-up, valid activation and release detection routines, and automatic shutdown processes. This phase represents the state of the *SenseModule* development as presented in this paper.

As with the *SMARTDOT* module, it must be noted that the author has been the sole user of the *SenseModule* to this point in its development, and that all of the raw data waveforms collected and presented in this paper were generated from the author's particular bowling style. Thus, the automatic detection algorithms currently implemented in the embedded software are undoubtedly biased towards that style. Careful attention has been applied in attempting to design generic and robust detection algorithms, but further extensive data collection and testing across multiple bowling styles must be

conducted, followed by additional analysis and refinement of the resulting raw data waveforms, in order to insure truly robust and reliable autonomous *SenseModule* operation across a variety of users.

Appendix B (page 117) presents a side-by-side comparison of typical ambient light and impact data recorded by the original *SMARTDOT* module and the *SenseModule*. Appendix E (page 125) includes graphs of the raw data waveforms from the 18 Ball Records that were collected during the last testing session of the *SenseModule* that utilized the hardware design and embedded software presented in this paper.

4.1 Physical Constraints

The *SenseModule* complies with all of the previously specified physical design constraints (2.1.1):

1) **Transparent:** Operation of the *SenseModule* requires no intervention on the part of the bowler, and it is not physically intrusive under the finger insert. The *SM* automatically awakens from **SleepMode** when the bowler picks up the ball and puts their fingers in the finger holes, and records the sensor data starting with the bowler's approach through the ball falling into the pit at the end of the lane. When the bowler wants to upload the data, they place the *ComModule* over the finger hole containing the *SM*, and the data is automatically transferred to the *CM*.

2) **Small and Light Weight:** The dimensions and weight of the *SenseModule* prototype are given in Table 13 below. The *SM* prototypes were built on 1.60 mm (0.062") thick PCBs, but 0.080 mm (0.031") thick PCBs could be used, reducing the weight by ~1.00 gm. Using a CR2016 90 mAh battery would reduce the height and weight by 1.6 mm (0.063") and 1.75 gm (0.044 oz), respectively, but would also yield only 35% of the battery life. The dimensions and weight do not include a plastic holder for the PCB and battery.

Table 13: *SenseModule* Dimensions

SenseModule	**As Built**	**As Built w/CR2032 225 mAh Battery**	**0.080 mm PCB w/CR2032 225 mAh Battery**	**0.080 mm PCB w/CR2016 90 mAh Battery**
Diameter	24.2 mm (0.951")	-same -	-same -	- same -
Height	4.8 mm (0.190")	8.0 mm (0.315")	7.2 mm (0.285")	5.6 mm (0.220")
Weight	2.00 gm (0.071 oz)	5.25 gm (0.185 oz)	4.25 gm (0.150 oz)	2.50 gm (0.088 oz)

3) **Low Cost:** The component cost for the *SenseModule* comes in under $15 in quantity (10,000 pieces), including the battery, but excluding an as-yet-to-be-developed plastic case.

4) **Low Power:** The average current draw of the *SenseModule*'s major modes of operation is given in Table 14. The values in the "Current" column are calculated based on the component values and data sheet properties. The totals at the bottom of the table are measured RMS values.

   Over the course of a year, the *SenseModule*'s quiescent **SleepMode** current of 2.5 μA drains about 22 mAh from the 225 mAh CR2032 battery, leaving 203 mAh for normal operation. Average accumulated run time for each Ball Record appears to be about 30 seconds, including false activations during the trip back from the pinsetter to the ball return. Thus, a single true activation results in a total draw of 40 mAs for the battery, yielding 20,000 such activations per battery in a year. If we assume an average of 18 activations per game, a single CR2032 battery will yield 1100 games per year. If we further assume that only half of the battery capacity is

usable in this application, the *SenseModule* is still capable of recording 550 games on a single CR2032 battery. Using a 90 mAh CR2016 battery will reduce the overall height and weight of the module significantly, but will only yield 35% of the above estimate, or ~200 games.

Table 14: *SenseModule* Current Draw

SenseModule	Current (ave)	SleepMode	CommandMode (5-10 s)	ApproachMode (10-30 s)	SampleMode (<= 5 s)
Startup Circuit	1.3 µA	X			
CP0	0.5 µA	X			
smaRTClock (RTC)	0.6 µA	X			
8051F921 (µP)	600 µA		X	X	X
TSL13	50 µA		X	X	X
ADXL345 (sample)	180 µA			X	X
ADXL345 (read)	100 µA			X	X
EEPROM (write)	115 µA				X
EEPROM (read)	100 µA		X		
TRX LED (IREF0)	250 µA		X		
Average Current		2.5 µA	1.10 mA	1.03 mA	1.15 mA

4.2 *SenseModule* Hardware

Pictures of the current version of the SenseModule prototype are shown in Figure 17. The prototype is shown true size, with the top of the *SenseModule* on the left and the bottom on the right.

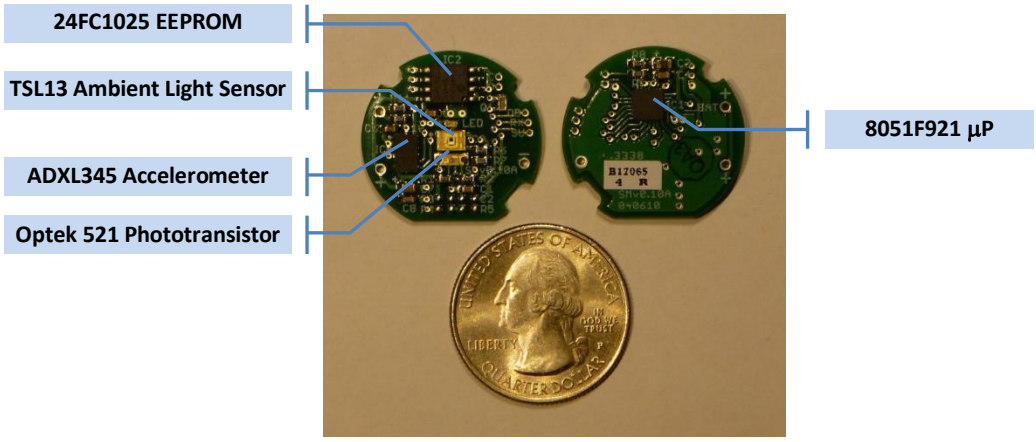

Figure 17: *SenseModule* Prototype

4.2.1 Start-Up Circuit

The ambient light-based startup circuit has proven sufficient for this application. It ignores transient light/dark pulses, and awakens the *SenseModule* from **SleepMode** only after a transition to an extended period (~500 ms) of near total darkness. That condition generally occurs in response to the following:

1) Normal activation before the bowler's approach
2) Normal activation for uploading data via the *ComModule*
3) Pinsetter elevating the ball from the pit to the subway ramp
4) Ball entering the subway at the pinsetter
5) Ball exiting the subway at the ball return
6) Ball rolling on the ball return
7) Ball being placed in a bag, locker, trunk, or closet

Conditions 3-7 are considered false activations. However, each of those occurrences yields only a single activation, after which, the *SenseModule* returns to **SleepMode** and must again be exposed to light, and then an extended period of darkness before the next activation can occur. There are additional steps that could be implemented in software to detect those false activations, e.g. checking the ADXL345 for a certain threshold of motion as part of the wake up validation checks.

4.2.2 Microprocessor (8051F921)

The 8051F921 µP has proven to have all the functionality and versatility necessary to meet the evolving requirements of this application. Besides the original reasoning for selecting this µP for the *SenseModule*, the following functionality has also proven to be quite useful:

1) The crossbar switch enabled versatile function assignment to port pins, which greatly simplified the PCB layout.
2) The ability to use a configurable on-board comparator as a wakeup source eliminated the need for an external D-FF that had originally been used in the startup circuit, and enabled the use of the *smaRTClock*, since waking up from **SleepMode** does not require resetting the *smaRTClock*.
3) The port match interrupt provided a versatile interface to the interrupt pins of the ADXL345.
4) The *smaRTClock* (RTC) was not originally considered for the design, but has proven to be useful in several ways: it provides an accurate time base for both sensors (which operate at different sampling frequencies), it provides a time base that enables time/date stamping of the Ball Records, and it can also be used to track run time, and thus battery usage.
5) The built-in CRC function is also convenient for calculating the CRCs for the EEPROM pages, as well as for the reliable exchange of data between the *SenseModule* and the *ComModule*.
6) The built-in programmable current source (IREF0) drives the transmit LED, allowing for configurable control of the LED current, eliminating the need for a current limiting resistor.

4.2.3 EEPROM (24FC1025)

The 128-byte page size of the EEPROM, along with the ability to buffer and write 128-bytes during a single transaction has proven to be quite valuable in this application. The page buffering allows for consolidating and optimizing I^2C bus activity, while also minimizing the number of EEPROM write cycles, which reduces the overall current drain on the battery during **SampleMode**.

4.2.4 Accelerometer (ADXL345)

This design leverages the autonomous sampling capabilities of the ADXL345. The ADXL345 has its own configurable sample timer, contains an internal FIFO sample buffer, and supplies interrupts to the µP when that buffer is full. The *SenseModule* leverages those capabilities so that the ADXL345 can accumulate samples on its own, while the µP concentrates on sampling the ambient light waveform, and on transferring light and ADXL pages via the I^2C bus. With the ADXL345 configured for autonomous operation, the µP is able to spend more time in its internal low-power **IdleMode**, which reduces the overall *SenseModule* current during both **ApproachMode** and **SampleMode**.

Analog Devices has recently released a lower cost accelerometer (ADXL343) that is pin-for-pin, and functionally compatible with the ADXL345. The cheaper ADXL343 has a lower acceleration bandwidth of 320 Hz, but that lower bandwidth is actually a plus in this application, as it will serve as a "pre-filter" for the higher frequency vibration and noise components that the ADXL345 can detect.

4.2.5 Ambient Light Sensor (TSL13)

The TSL13 light-to-voltage converter is admittedly overkill for this application. It was included to enable the *SenseModule* to collect light waveform data similar to that collected by the original *SMARTDOT* module, as well as to correlate the ambient light readings with the ADXL345's acceleration readings.

Given that the TSL13 was already designed into this version of the *SenseModule*, it also doubles as the infrared receiver for the *SenseModule*. However, its relatively slow step response limits the *SM* to receiving data at 14.4 to 28.8 Kbaud. Although those rates are sufficient for receiving commands from the *ComModule*, which are relatively short, the infrared UART will eventually be used to transfer embedded software updates to the *SenseModule*, which will be 16 Kb to 32 Kb, and could take 10-20 seconds to transfer at those slower baud rates.

Now that the ambient light data has been collected in conjunction with the 3-axis acceleration data, the TSL13 can be replaced with a less expensive and more power-efficient circuit. In fact, it is possible that a single Optek 521 phototransistor could be used for the startup circuit, the ambient light sensor, and the infrared receiver by having the µP switch in/out appropriate biasing resistors, and configure its internal comparators, as needed.

4.2.6 Infrared Transmitter (IREF0 and LED)

The programmable constant current source (IREF0), combined with a high efficiency LED, provides consistent drive current and intensity at sub-milliamp current levels, and the switching speed is fast enough for UART operation up to 115/230 Kbaud.

4.3 Raw Data Waveforms

The *SenseModule* collects four channels of sensor data: ambient light from the TSL13 light-to-voltage converter, and the three acceleration axes from the ADXL345 accelerometer. A graph of the four channels of raw data contained in a typical Ball Record is shown in Figure 18. The time axis across the bottom starts when the *SenseModule* entered **ApproachMode** and runs until the *SM* detected an automatic shutdown condition, which terminated **SampleMode**. The dead space on the graph before 8.25 seconds is the time the *SM* spent in **ApproachMode** waiting for release, storing sample pages in the circular buffers that were eventually overwritten by more recent data. The vertical axis is in G's, where 1 G = 9.8 m/s^2 (acceleration due to gravity). The ADXL345 has a range of ± 16 G's, and the TSL13 has an output range of 0-255 counts, which has been scaled to 0-16 G's on the graph.

- **Result:** Indicates the scoring result of the ball. In this case, the ball entered high in the pocket, and the '4' and '7' pins were left standing after the first ball.
- **Time:** Time of day and date that the *SenseModule* was activated ('0' on the time scale), taken from the µP's *smaRTClock*.
- ***T_s*:** Total sampling time, in seconds, for this activation of the *SenseModule*.
- **ADXL *F_s*:** Effective sampling frequency of the ADXL345 for this Ball Record, as measured against the µP's *smaRTClock* (RTC). The ADXL345's internal sample clock is configured for 200 Hz.
- **Light *F_s*:** Effective sampling frequency of the TSL13 for this Ball Record, as measured against the µP's *smaRTClock* (RTC). The µP generates a 240 Hz clock for (over)sampling the TSL13 output.

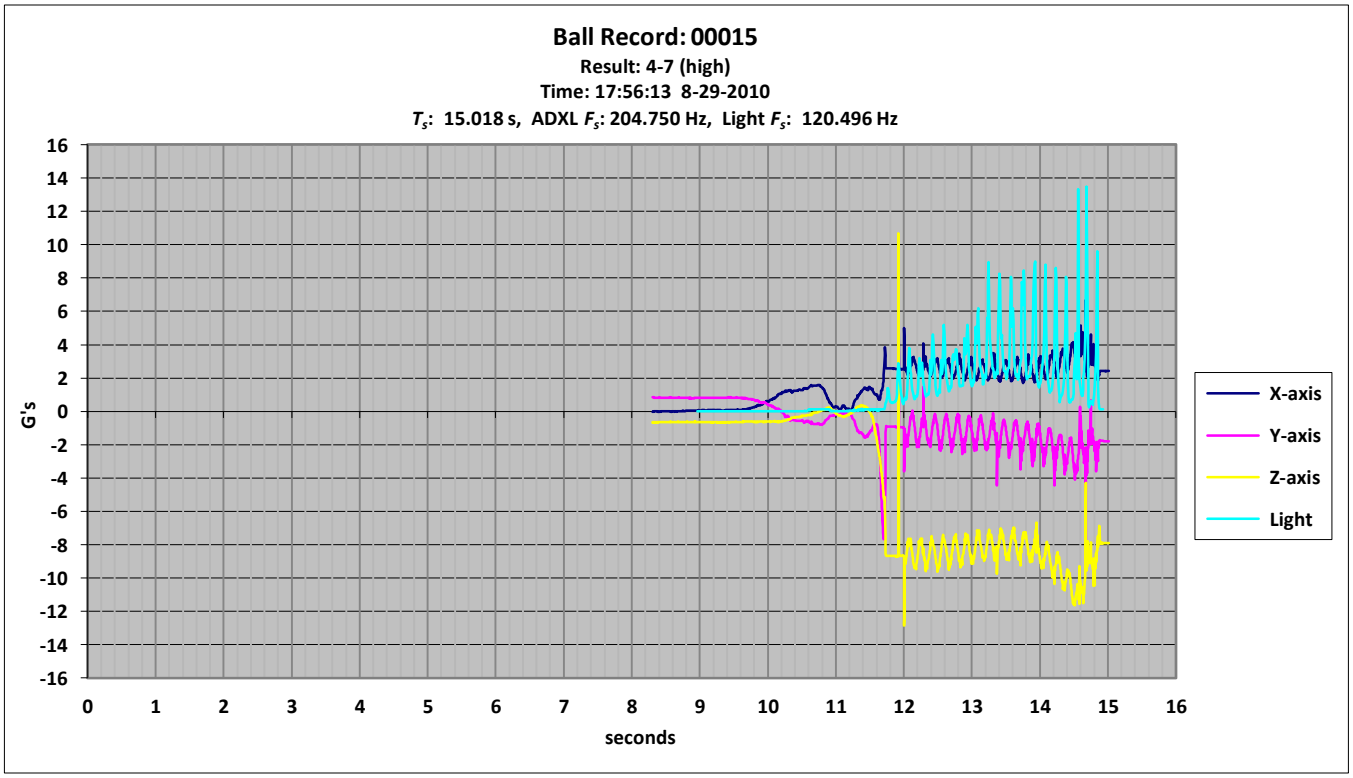

Figure 18: Typical Raw Data Waveform

4.3.1 Typical Waveform Regions

Figure 19 is zoomed in on the typical waveform of Figure 18. The graph is split into the portions of the waveform that were captured in **ApproachMode** and **SampleMode**, respectively. The various regions of interest are demarcated in red. In chronological order, those regions are:

1) **Stance:** The bowler and ball are relatively motionless, as the bowler prepares to start their approach. The bowler's fingers block the ambient light from reaching the TSL13.
2) **Approach:** The bowler starts their approach by leaning, taking a step, and/or pushing the ball forward. The motion of the ball from the bowler's arm swing is apparent. The bowler's fingers still block the ambient light from reaching the TSL13.
3) **Release:** The bowler starts applying lift to the ball shortly before release, inducing rapid acceleration in the *Y* and *Z* axes. The light level increases as the bowler's fingers leave the ball.
4) **Loft:** The bowler has released the ball, and it is initially in free fall, thus the flat acceleration lines. The ball bounces twice as it hits the lane, as shown by the sudden spikes in all three axes.
5) **Reaction:** The ball rolls down the lane, and the tilt sensing aspect of the ADXL345 is evident in the roughly sinusoidal waveform as the *SenseModule* rotates through the gravitational field. The light waveform also indicates rotation, from release all the way through the Impact Region.
6) **Impact:** The ball hits the pins, as indicated by the increased "noise" and the spikes in the 3 axes. The light level peaks as the ball passes under the fluorescent light illuminating the pins.
7) **Shutdown:** The ball falls off the back of the pin deck, into the pit. The ADXL axis waveforms once again flatten out, due to free fall, and the ambient light level drops to near 0.

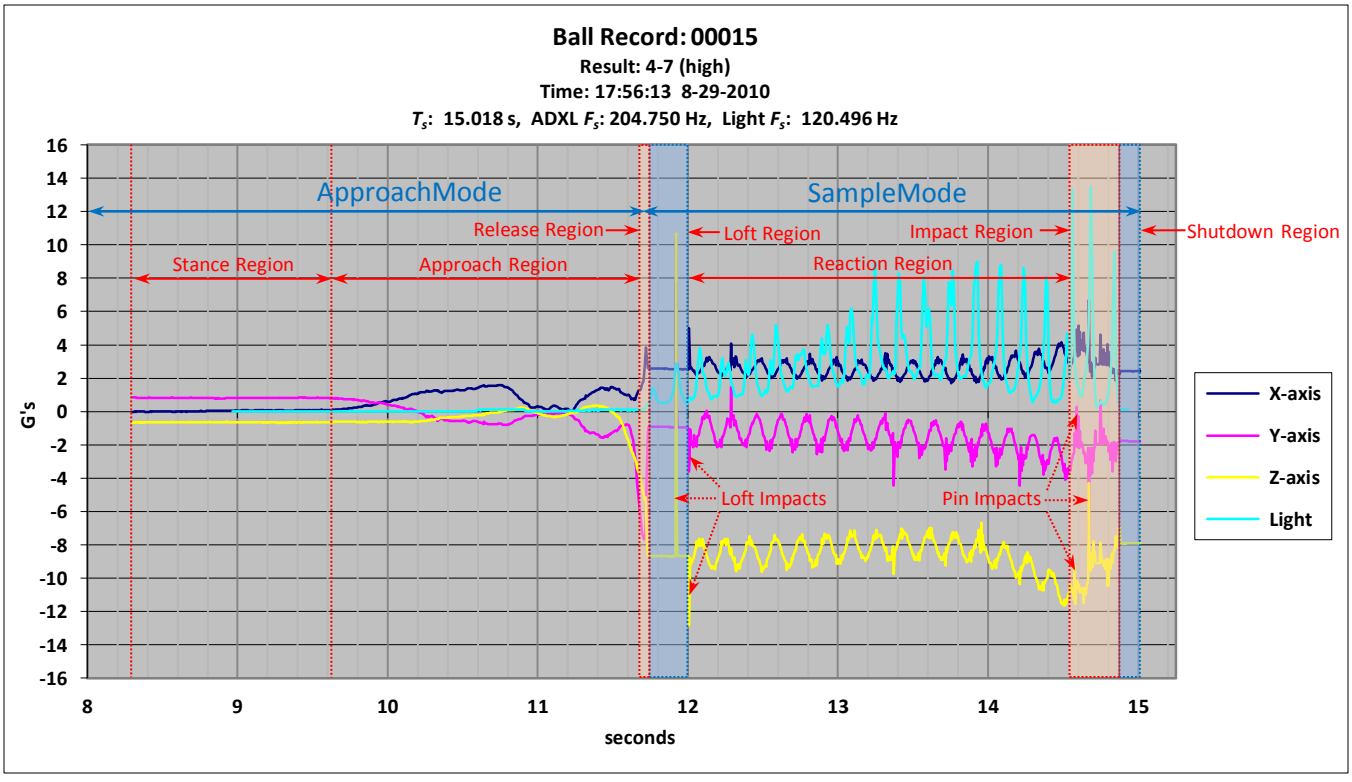

Figure 19: Raw Data Waveform Regions

4.3.2 False Activation Waveforms

The *SenseModule* also wakes up from "false" activations generated during the ball's trip from the pinsetter back to the ball return. Figure 20 displays a graph of the resulting data. The *SenseModule* had just recorded a legitimate Ball Record, and returned to **SleepMode**. While the ball was in the pinsetter, the *SM* experienced a prolonged dark period, sufficient to wake up the *SM* again. During **WakeUpMode**, the *SM* detected a sufficient level of darkness to enter **ApproachMode**, and advanced to **SampleMode** when the ball emerged into the light at *A*. **SampleMode** timed out at the end of *E*.

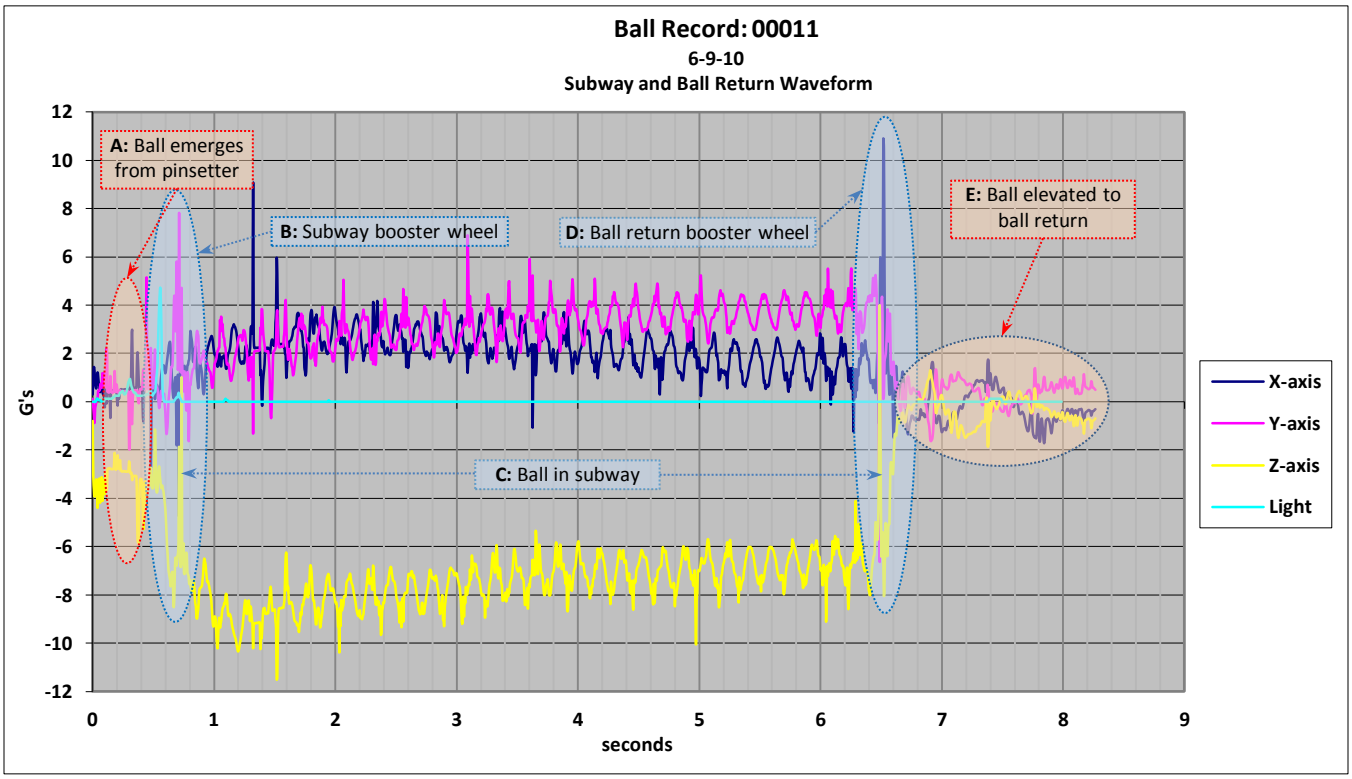

Figure 20: Subway (False) Activation

A) The pinsetter picked up the ball, and placed it on the subway acceleration ramp. As the ball emerges from the pinsetter, the light level increases and the *SenseModule* detects release.

B) The ball encounters the subway booster wheel, which accelerates the ball and sends it into the subway. There is another spike in the light level from the rotation of the ball, as well as spikes in the ADXL axes from the sudden acceleration applied by the ball wheel. There is also a sustained increase in the ball's angular velocity, indicated by the amount the *Z*-axis is offset from 0 G's.

C) The ball rolls along the subway, as noted by the tilt response signatures of all three axes, the absence of ambient light, and the continued offset of the *Z*-axis from 0 G's.

D) The ball reaches the ball return booster wheel, which elevates the ball from the subway to the ball return. The acceleration spikes when the ball encounters the ball wheel. Ambient light is still absent, as the ball has not yet emerged from the subway.

E) The ball transitions from a rapid rotation to a much slower rotation as it is elevated to the ball return. The ball still has not emerged from the subway, as indicated by the absence of light.

The waveform shown above in Figure 20 is the result of the *SenseModule* waking up while in the pinsetter, and detecting release soon after entering **ApproachMode**, as the ball emerged into the light before entering the subway. Figure 21 depicts a second form of false activation. In this scenario, the *SM* was in **SleepMode** when it emerged from the pinsetter. The ball entered the subway, woke up after 500 ms of darkness, entered **ApproachMode**, and waited for release. The **ApproachMode** time out period (30 seconds) is sufficient for the ball to traverse the subway and emerge from the ball return, where the *SM* detected release, and transitioned to **SampleMode**.

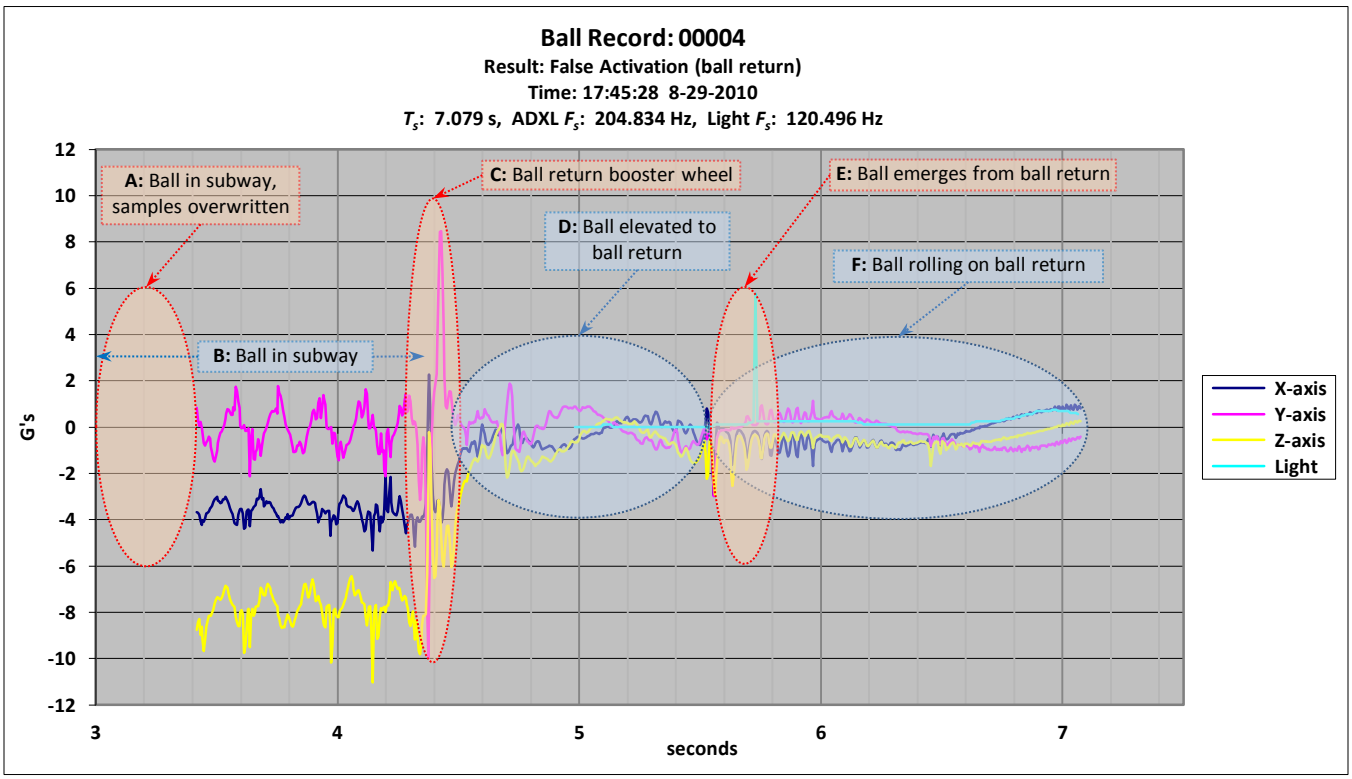

Figure 21: Ball Return (False) Activation

A) The ball is in the subway. The first 3.4 seconds of sensor readings were overwritten in the *SM*'s circular page buffers while it was waiting for release during **ApproachMode**.

B) The ball is still in the subway. In **ApproachMode**, the *SenseModule* captures up to 3 seconds of the sensor readings that immediately precede release in its circular page buffers.

C) The ball reaches the ball return booster wheel, as indicated by the sudden spikes in acceleration and the cessation of rotation.

D) The ball is elevated to the ball return, as indicated by the slow rotation. The *SM* is still in the dark, with the ball having not yet emerged from the ball return.

E) The light level increases as the ball emerges from the ball return at 5.6 seconds. The *SenseModule* detects release at 5.7 seconds, as the finger hole with the *SM* rotates toward the ceiling.

F) The ball is slowly rolling around the ball return, while also wobbling from side to side, as indicated by the high frequency, low amplitude spikes in the acceleration waveforms. The *SM* shuts down automatically due to the extended period of relative inactivity on all sensor channels.

4.4 Automatic Functions

In order for the *SenseModule* to operate without any user intervention, it must be able to automatically identify and discriminate between the various regions described in Section 4.3. It must also be able to discriminate between the legitimate activation shown in Figure 19, and the false activations shown in Section 4.3.1. The critical task is to identify and recognize regular patterns across the raw data channels, and then implement efficient algorithms so that the *SenseModule* can automatically discern the presence (or absence) of those patterns. Looking at the raw data waveforms shown in Figure 19, that task might seem relatively straight-forward, as there are clear demarcations in the waveforms between those regions. However, the situations depicted in Figure 20 and Figure 21 occur regularly, and result in waveforms with very similar characteristics. How to discriminate between those types of waveforms, and reliably detect valid activations while ignoring false activations, is not so obvious.

The task becomes even more complex when constrained to working with an 8-bit processor, running at 3.05 MHz, with internal memory access limited to 4.25 Kb of RAM. In addition, the *SenseModule*'s collection of automated detection algorithms must operate in real-time, in order to limit the μP's run time to collecting and storing legitimate waveforms. Several automatic functions were initially developed and further refined through multiple iterations of data collection, analysis, and tweaking of the embedded software. Much of the initial valid waveform and release detection development/testing/refinement was conducted in the author's basement, before a ball was ever rolled down a real bowling lane. Several real-world testing sessions then followed at a local bowing establishment, interspersed with more "basement bowling" sessions. The automatic detection algorithms have been further refined since the testing for this paper was conducted. The updated algorithms are proprietary at this time, thus they are not presented in this paper.

4.4.1 Valid Activation Detection

A normal waveform is the result of the bowler retrieving the ball from the ball return, placing their fingers in the ball (inducing *SenseModule* start-up), and then delivering the ball to the lane, with a characteristic sudden increase in acceleration, immediately followed by an increasing light level.

A typical **ApproachMode** waveform is shown in Figure 22. The components of a valid activation are:

- **A) ApproachMode:** Extends from time 0, when the bowler first put their fingers in the ball, blocking light to the *SenseModule*, and continues to a point within the Release Window (*E*).
- **B) Stance:** Bowler and ball are relatively motionless, as the bowler prepares for their approach.
- **C) Approach Motion:** Encompasses all motion from the start of the bowler's approach to release of the ball. The bowler leans forward, takes a step, and/or pushes the ball away from their body to start their approach. During the bowler's arm swing, the ball arcs forward, then back, and then forward again, followed by the bowler applying lift to the ball (*D*) just before release (*E*).
- **D) Lift Motion:** Indicated by extended acceleration, primarily in the *Y* and *Z* axes. The *Y*-axis points in the general direction of the lane and the *Z*-axis points toward the center of the ball.
- **E) Release Window:** Release occurs at the end of lift, marked by the flattening of the acceleration curves (ball is in free fall during loft), and a marked increase in the ambient light level.

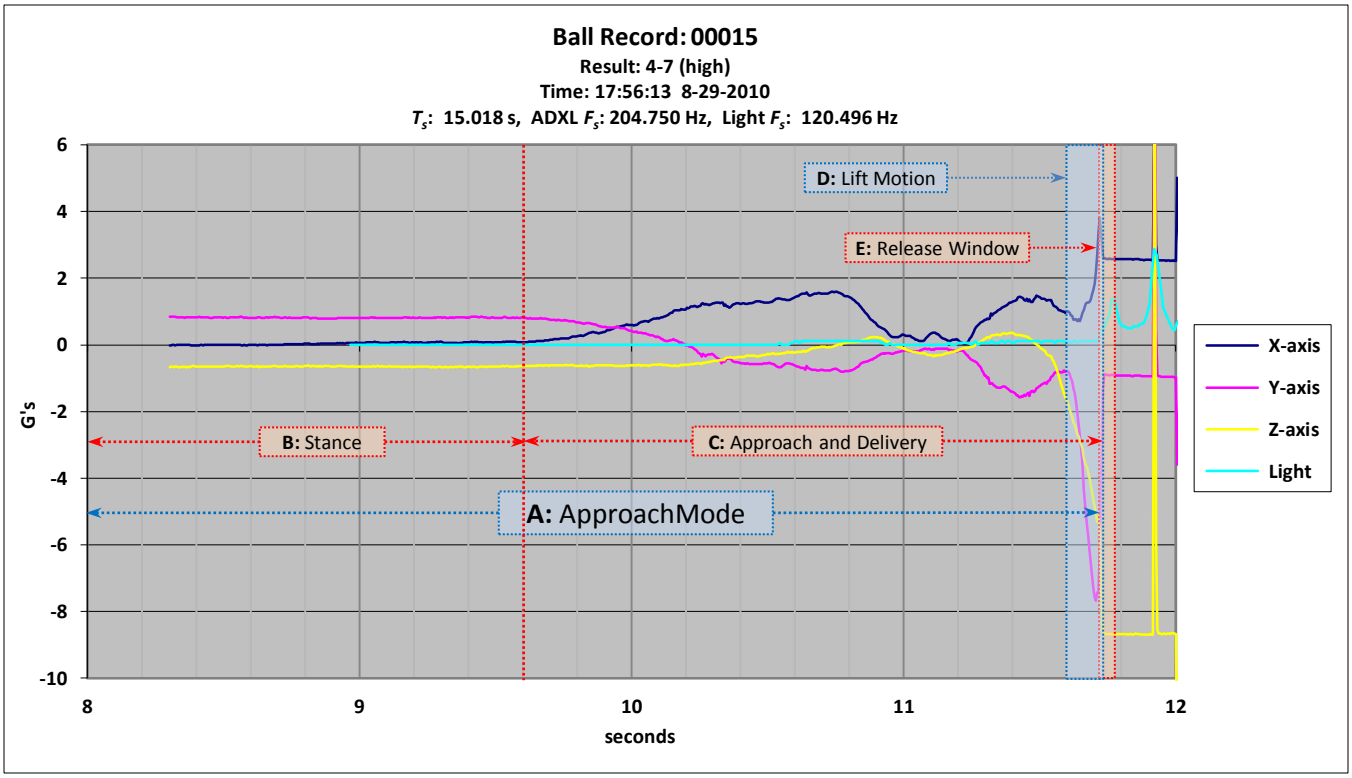

Figure 22: Expanded ApproachMode Waveform

In order to extend battery life, and maximize the number of valid waveforms that can be stored in EEPROM, the *SenseModule* must be able to quickly discriminate between valid activations generated by the bowler delivering the ball, and the regular "false" activations that result from the pinsetting machine sending the ball into the subway and back to the ball return.

Since the *SenseModule* can spend up to 30 seconds in **ApproachMode** waiting for release, it also performs valid activation detection during that time. While the *SenseModule* waits for the bowler to deliver the ball, it constantly evaluates the incoming sensor readings for light and motion consistent with the bowler delivering the ball to the lane. As the *SM* records that sensor data in its circular page buffers, it also looks for evidence of false activation. When the discrimination algorithm identifies sensor readings consistent with a false activation, the *SenseModule* immediately exits **ApproachMode**, and returns directly to **SleepMode**, without transferring any sensor data to the EEPROM, just as it does when **ApproachMode** times out.

During development and testing, the *SenseModule* recorded numerous false activation scenarios, two of which are presented in Section 4.3.1. When the ball falls off the end of the lane, and into the pit, it generally enters a period of prolonged darkness. Once the ball is in the pit, the pinsetting machine elevates the ball to the subway ramp by means of a spinning wheel or rotating belt that accelerates the ball, while also inducing the ball to rotate. During the ball's transfer from pit to subway, it is possible for the *SM* to sense an ambient light start-up condition, followed by increasing acceleration levels while still dark, followed by an increasing light level, with the ball then rolling along the subway rails.

If the *SenseModule* hasn't woken up before entering the subway, it can still wake up as a result of that event, since the *SM* once again is exposed to a prolonged period of darkness in the subway before emerging back into the light on the ball return, where conditions of increasing acceleration followed by an increasing light level can again occur.

Thus, normal activation cannot be uniquely described by a simple scenario of prolonged darkness, with an eventual sudden increase in acceleration on multiple axes, followed by an increasing light level. Further complicating the matter of reliably detecting false activations is that the *SM* must err on the side of caution, e.g. it must detect every valid wake up, at the expense of allowing some false activations to continue into release detection.

The current version of the false activation detection algorithm considers the following conditions:

1) The magnitude, frequency, and rate of change of the ambient light level
2) The magnitude, frequency, and rate of change of the acceleration axes
3) The relative phase of the changes in light level and acceleration across all three axes
4) Various timing constraints between certain light and acceleration events

The results of the false activation detection routine tested for this paper have been mixed. Table 15 summarizes the results from testing of the *SenseModule* routine. Across 20 games (200+ frames) of testing, the *SenseModule* captured the vast majority (over 97%) of true activations, but it rejected only 68% of the false activations. Even at 97%, the miss rate on valid activations is still too high; perhaps 1 miss in 1000 valid activations is acceptable. It is also apparent that there is a high rate of false activation (over 80%) induced by the pinsetter/subway/ball return. Fortunately, there is another line of "defense" in rejecting false activations, which is implemented at the release detection level.

Table 15: False Activation Detection

SenseModule **(219 frames)**	**Valid Activations**	**False Activations**
Captured	214	56
Rejected	5	123
Total Events	219	178
Detection Efficiency (%)	97.7%	68.5%

4.4.2 Valid Release Detection

The *SenseModule* remains in **ApproachMode** for up to 30 seconds, waiting for the bowler to release the ball. When the *SM* detects release, it transitions from **ApproachMode** to **SampleMode**, and begins committing the circular page buffer contents collected during **ApproachMode** to EEPROM, as part of a new Ball Record. The *SM* needs to detect release in a timely fashion so that it can begin transferring the **ApproachMode** page buffer contents to EEPROM before that data is overwritten by the new sensor data collected during **SampleMode**.

A typical release scenario is shown in Figure 23. The components of a valid release are:

A) ApproachMode: Starts at time 0 and ends with transition to **SampleMode** after release detection. The *SenseModule* stores sensor data in the circular page buffers during this mode.

B) SampleMode: The transition from **ApproachMode** occurs upon the *SenseModule* detecting release. The *SM* commits circular buffer pages to EEPROM during this mode.

C) Approach Region: Begins with bowler's approach. Ends with delivery of the ball at release (*H*).

D) Loft Region: Starts immediately after release (*H*). Contains multiple impacts with the lane (*J*).

E) Rotation Region: The ball is in constant contact with the lane, rolling toward the pins.

F) Lift Motion: The bowler imparts lift to the ball, as indicated by increased acceleration in the Y and *Z* axes waveforms, which continues until release, when the waveforms flatten out.

G) Release Window: The *SenseModule* must detect release within this window, which is bounded by an increasing light level and sudden changes in acceleration on all three axes.

H) Release Point: The true release point, where the ball enters free fall (*I*), as indicated by the sustained increased light level and the flattening of the acceleration waveforms.

I) Free Fall: The flat acceleration waveforms indicate the ball was released above the level of the lane. It is in free fall until it impacts the lane at (*J*). Not all bowlers loft the ball this much.

J) Loft Impacts: The ball bounces off the lane after the first impact, but remains in contact with the lane after the second impact, which starts the rotation region at (*E*).

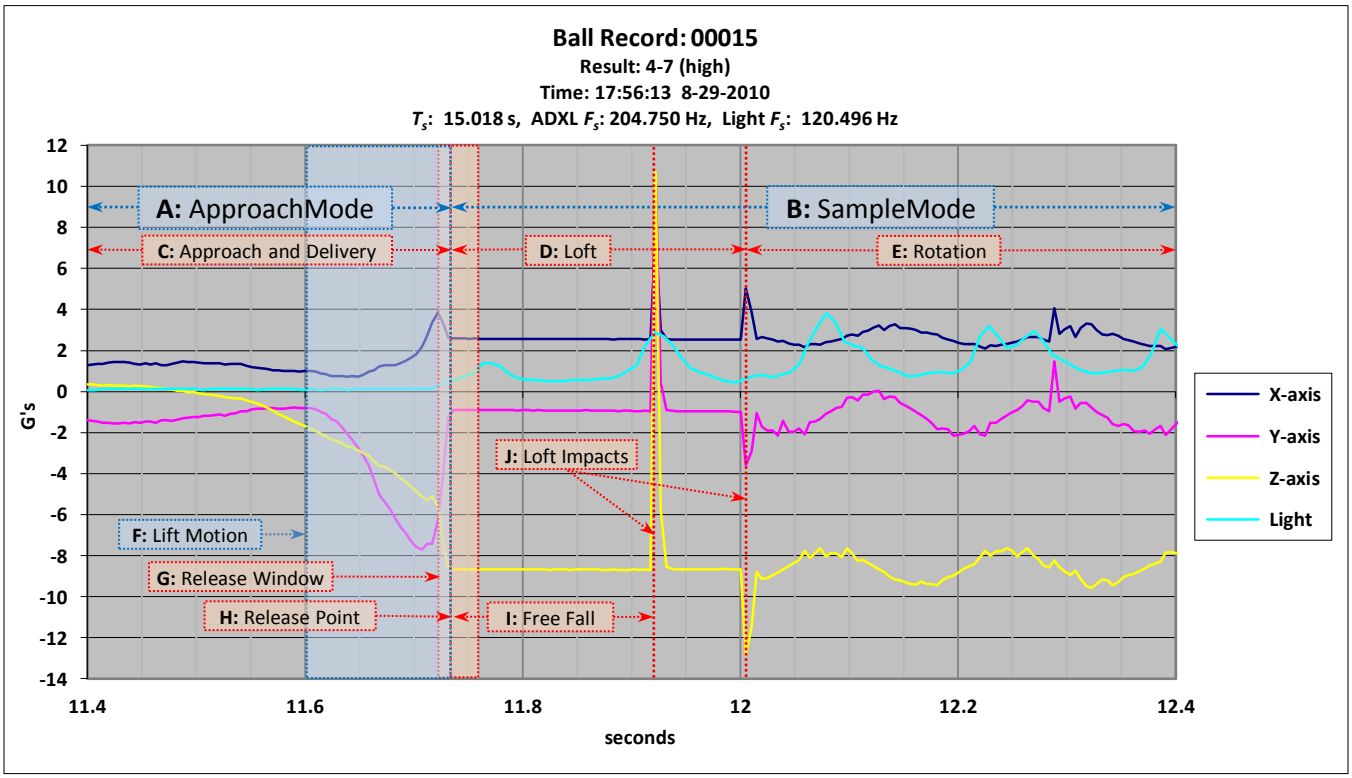

Figure 23: Expanded Release Region

The release detection scheme might seem obvious from Figure 23: identify the increasing *Y* and *Z* axes acceleration during the absence of light, and then look for an increasing light level, followed by the flattening of the acceleration curves. However, this graph is taken from one frame from one bowler. Not all bowlers apply as much lift to the ball, nor do they apply lift with the same timing, nor do they loft the ball as much. In fact, many avid bowlers deliver the ball very smoothly to the lane, with little or no loft, and with much less lift. Complicating matters, an individual bowler may alter their release and loft in order to adjust to varying lane conditions. Thus, the seemingly obvious release indicators in the above graph will not be so obvious (and in some cases likely to be absent) for many bowlers.

Recalling the two false activation scenarios presented in Section 4.3.1, it is possible that if either of those scenarios is not rejected as a false activation during **ApproachMode**, they may each eventually lead to false release events. Both scenarios are followed by the ball emerging from darkness into light, while the ball is also experiencing sudden changes in acceleration. It is the eventual detection of a release event that leads to the *SenseModule* recording a false activation as a Ball Record in EEPROM, likely overwriting a legitimate Ball Record with one from a false activation. Thus, there is also a need for a false release detection algorithm.

The false activation and release detection routines run simultaneously during **ApproachMode**. If a false activation is detected first, release detection is aborted (along with **ApproachMode**), and the *SenseModule* returns to **SleepMode**. If release is detected first, the false activation routine is aborted, and the *SM* transitions to **SampleMode**.

In order to reliably detect true release conditions, while rejecting releases that occur as part of a missed false activation, the *SenseModule* first identifies a likely release event, switches to **SampleMode**, and begins recording data to the new Ball Record in EEPROM. The false release detection routine then constantly evaluates the sensor readings following the initial release to determine whether or not to continue **SampleMode**. False release detection can take several hundred milliseconds to decide on a result. If the *SM* does detect a false release, it aborts *SampleMode* without advancing the *nextBallRecord* and *nextBallPage* pointers. Thus, upon the next valid activation and release, the *SM* will overwrite the partial data it previously recorded from the false activation and release with a valid new Ball Record.

As with false activation detection, false release detection must be conservative, e.g., it must favor valid releases, at the expense of missing some false release events. The current version of the false release detection algorithm considers similar factors as false activation detection, but applies different emphasis and priority to those factors. In essence, false release detection is an extension of false activation detection, just as **SampleMode** is an extension of **ApproachMode**.

The results of the implementation of the false release detection routine tested for this paper have also been mixed. Table 16 summarizes the results of the false release detection algorithm. The *SenseModule* keeps track of the number of times it rejects (ignores) false activations and false releases. Although the *SM* doesn't know how many false activations/releases it fails to catch, any false activation that makes it through both detection routines results in the waveform eventually being stored as a Ball

Record in EEPROM. Such records are easy to recognize visually after the data has been uploaded to the *ComModule*. Thus, it is possible to discern between false activations caused by the pinsetter and those caused by the subway/ball return that made it through both detection routines.

Table 16: False Release Detection Results

SenseModule (219 frames)	Valid Releases	Pinsetter "False" Releases	Subway/Ball Return "False" Releases
Captured	214	2	13
Rejected	5	41	
Total Events	219	56	
Detection Efficiency (%)	97.7%	73.2%	

Of the 56 total false activations that got through to false release detection, 41 were rejected by the routine, yielding a combined detection efficiency of 93%. That level must certainly be improved upon, but it is encouraging given the limited level of scrutiny and analysis that was applied to the sensor data in developing the detection algorithms used for testing purposes for this paper.

4.4.3 Shutdown Detection

If it were not for the dual concerns for minimizing *SenseModule* run time, while maximizing the Ball Record capacity of the EEPROM, the *SenseModule* could simply record a fixed amount of data for each Ball Record. However, given those two goals, the *SenseModule* should stop recording sensor data as soon as it has passed through the pins. Thus, an automated early shutdown algorithm has also been implemented. Figure 24 shows a graph of a typical impact region.

A) **SampleMode:** The *SenseModule* commits sensor data to a new Ball Record in EEPROM.
B) **SleepMode:** The *SenseModule* returns to **SleepMode** after shutdown is detected.
C) **Rotation Region:** The ball rotates as it rolls towards the pins.
D) **Impact Region:** The ball hits the pins as it travels through the pin deck.
E) **Shutdown Region:** The ball falls off the back of the pin deck, and shutdown is detected here.
F) **Pin Deck Light:** The ambient light level spikes as the ball passes under the pin deck light.
G) **Pin Impacts:** The ball hits the pins, the *Z*-axis acceleration drops off, and all acceleration waveforms begin to exhibit increased noise levels.
H) **Free Fall:** The ball bounced in the air while hitting the pins, indicated by flat acceleration lines.
I) **Free Fall:** The ball is falling off the back of the pin deck. The ambient light level drops to near 0.

The automatic shutdown detection routine implemented in this version of the *SenseModule* is fairly straight forward. It relies on a combination of three basic characteristics of the raw data:

1) The three acceleration waveforms simultaneously flatten out as the ball goes into free fall off the back of the pin deck.
2) The ambient light level generally flattens out near 0 while the ball is in free fall.
3) Some combination of the above two elements persist for a minimum duration, which must be longer than the free fall period exhibited in (*H*) above. For the final version of the algorithm tested for this paper, the minimum persistence was set to 50 ms.

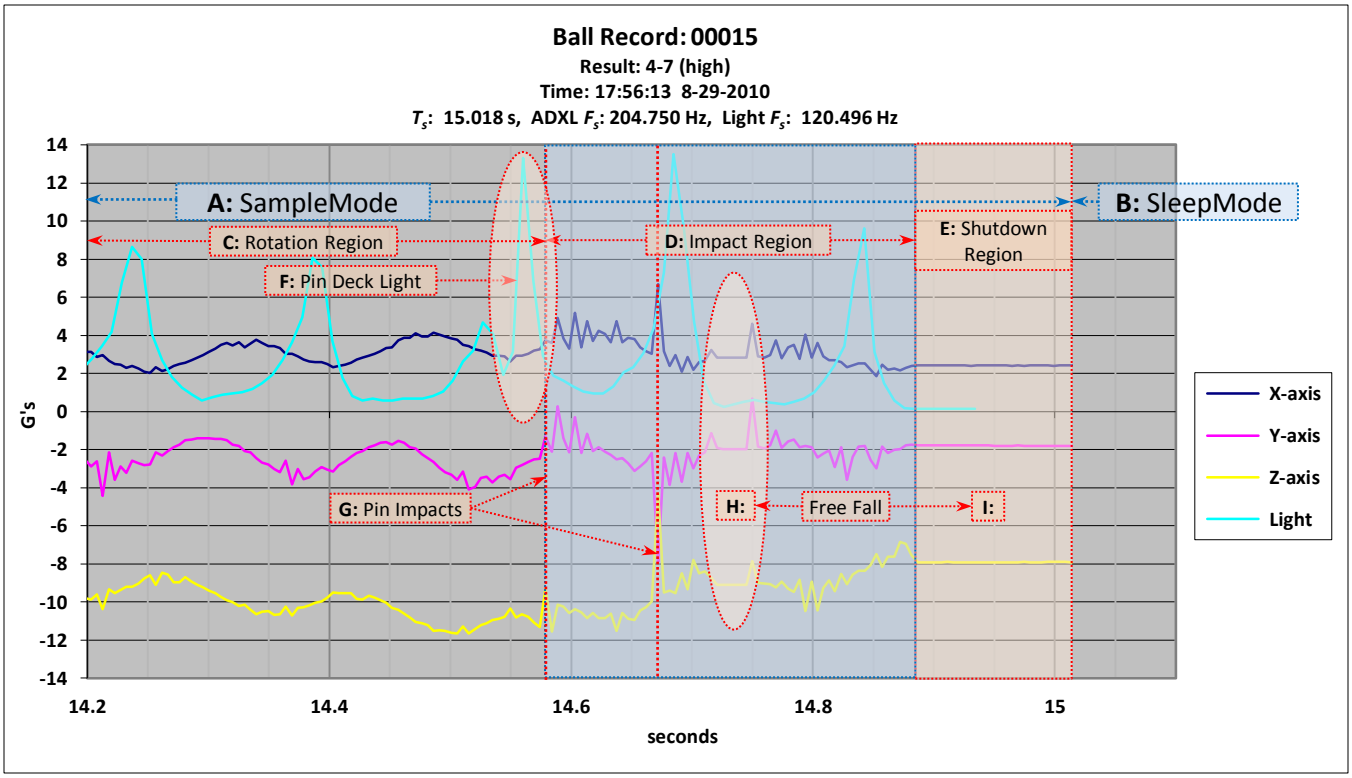

Figure 24: Impact and Shutdown Regions

There is also a shutdown time out period that starts at the beginning of *SampleMode* (upon release detection) that serves as a "fail-safe" in case early shutdown is not detected. *SampleMode* runs for a maximum of five seconds, and then automatically shuts down. The results for the shutdown detection algorithm developed for this paper have been quite promising. For all activations (valid and false) the routine detected early shutdown before the shutdown time out period expired every time (219 times out of 219 activations). Note that the *SenseModule* continues to collect samples until the current ADXL Sample Page is full, before it shuts down.

4.5 *SenseModule* Future Work

The initial development work has been accomplished, the basic waveform characteristics have been identified, and autonomous operation has been achieved. The next phase of development will be focused on refining the *SenseModule* hardware and embedded software to achieve more efficient and robust operation, as well as to reduce the cost, size, and weight of the module.

4.5.1 *SenseModule* Hardware

The *SenseModule* hardware design used for this paper is now four years old. Based on the knowledge gained through the collection and analysis of the raw data waveforms, the following refinements can be made to the *SM* hardware design.

1) The TSL13 light-to-voltage converter is no longer needed for this project. At a minimum, it can be replaced with a second Optek 521 phototransistor to serve as both the ambient light sensor, and as the serial receiver for the infrared UART. It might even be possible to multiplex those

functions with the existing Optek 521 by switching bias resistors, and changing comparator thresholds, as needed. Using the Optek 521 as the infrared receiver for the serial UART would also allow the use of a much faster baud rate (115/230 Kbaud) for the infrared UART's serial receiver. Consolidating the optical receivers into a single phototransistor is the cheapest solution for ambient light detection and serial reception.

2) Silicon Labs (the same company that makes the *SenseModule* μP) recently introduced the Si1141 ambient light and proximity sensor, which presents a highly promising and intriguing solution for replacing the *SM* start-up circuit and the TSL13 LTV. From the Silicon Labs website [29]:

> "The Si1141 is a low-power, reflectance-based, infrared proximity and ambient light sensor with I2C digital interface and programmable-event interrupt output. This touchless sensor IC includes an analog-to-digital converter, integrated high sensitivity visible and infrared photodiodes, digital signal processor, and an integrated infrared LED driver with fifteen selectable drive levels. The Si1141 offers excellent performance under a wide dynamic range and a variety of light sources including direct sunlight. The Si1141 can also work under dark glass covers. The photodiode response and associated digital conversion circuitry provide excellent immunity to artificial light flicker noise and natural light flutter noise. The Si1141 device is provided in a 10-lead 2x2 mm QFN package and is capable of operation from 1.71 to 3.6 V over the –40 to +85 °C temperature range."

The Si1141 in proximity sensing mode would not only replace the entire start-up circuit, it could also directly detect valid activations and release events, since it would only respond to the proximity of the bowler's finger inserted in the finger hole that contains the *SenseModule*. Such functionality would greatly reduce the time the *SM* spends out of **SleepMode**, since the pinsetter, subway, and ball return could no longer generate false activations.

The Si1141 would also serve as the ambient light sensor. Since the Si1141 can run autonomously, it would offload the burden of the entire ambient light sampling process from the μP, allowing the μP to spend more time in **IdleMode**.

The *SenseModule* would still require a phototransistor as the infrared receiver for its serial UART, since the Si1141 has neither an analog output, nor a step response fast enough for it to serve that purpose.

The added expense of the Si1141 would be offset by the cost of the components it replaces, combined with a significant reduction in the size, weight, and cost of the battery (CR2016 instead of a CR2032 3V lithium coin cell).

3) A plastic case must be developed for the *SenseModule*. The case will hold the *SM* and the battery, and allow for user-replacement of the battery. The case will have two pieces:

a. A holder that is permanently affixed at the bottom of the finger hole and facilitates accurate and repeatable positioning of the *SenseModule*.
b. A shell that encapsulates the *SenseModule* and the battery, and firmly secures the *SM*/battery assembly into the holder.

4) It should be possible to add an additional 24FC1025 EEPROM chip, which would double the capacity of the Ball Record database. The *SenseModule* currently has the capacity to store at

least 14 Ball Records (at the maximum 8.25 second sample time), but that is not enough to capture an entire game, which can consist of up to 21 individual rolls of the ball. Doubling the memory capacity of the *SM* would be enough to meet that goal. The Ball Record database, as it is currently designed, can manage up to 62 Ball Records without modification.

4.5.2 Embedded Software

The development of the *SenseModule* embedded software has been a major undertaking. The major thrust of this phase of the module's development was to create a fully functional, autonomous prototype. That goal has been achieved, but there is much more refinement of the embedded software to be accomplished.

1) As mentioned in Section 4.4, the automated detection functions require further refinement, e.g., they need to be more accurate, as well as more robust (accommodating of a variety of bowling styles). The inclusion of the Si1141 proximity sensor mentioned under the hardware section above would alleviate much of that effort, since it can directly detect the presence of the bowler's finger at activation, and its subsequent absence upon release of the ball. Absent that hardware (or something like it), the *SenseModule* software must continue to infer activation/release from the sensor data it currently collects. Increasing the reliability of those functions would quickly develop into a major research effort, as data from a variety of bowlers representing a cross-section of the major bowling styles would need to be collected and analyzed.

2) As currently configured, the *SenseModule* spends up to 30 seconds in **ApproachMode** collecting data while waiting for release, but then retains only the last three seconds worth of that data. The ADXL345 can be configured for autonomous motion sensing, in which the ADXL345 issues interrupt(s) to the μP upon detecting acceleration readings outside of configurable threshold ranges. After the μP has woken up and configured the ADXL345 for motion detection, it could return to **SleepMode** until the ADXL345 detects motion indicative of the bowler having started their approach, and issues an interrupt to wake up the μP to start **ApproachMode**.

3) The capacity of the Ball Record database could be further expanded through compression of the sensor data contained in the light and ADXL pages. The light and ADXL waveforms are frequently constrained to a small portion of the overall output range of their respective sensors. Offset and/or differential based-storage techniques could be used to compress the readings.

4) The current draw of the *SenseModule* is managed by using the μP's built-in **IdleMode**, and by enabling internal and external peripherals only when they are needed. Judicious switching of the μP's system clock while the μP is awake could further reduce the *SM*'s current draw.

5) The μP's flash program memory is writeable in-system. Currently, embedded software updates for the *SenseModule* prototypes must be delivered via a direct electrical connection. The ability to download software updates to the *SM* via its infrared serial UART must be added.

4.6 *SenseModule* Development Summary and Conclusions

The development of a small, low-cost, autonomous *in situ* sensor module that fits unobtrusively in an existing finger hole and collects, records, and transmits ambient light and 3-axis accelerometer readings has been achieved. The *SenseModule* fulfills all of the requirements and design criteria that were laid out earlier in the paper. As such, the *SenseModule* could be the first such device of its kind that meets those constraints. Although the development of the *SenseModule* has been a success, there is certainly room for improvement in the several areas listed in Section 4.5.

The development of a working *SenseModule*, along with the collection of the desired sensor data, represents the completion of the first part of this project. A visual inspection of the collected waveforms reveals the following notable characteristics that could be further exploited within the *SM* embedded software to enable more consistent autonomous operation.

1) The bowler's approach and arm swing generate a relatively slow, smooth, noise-free waveform which concludes with a sudden extended increase in acceleration due to the lift and turn the bowler applies to the ball just before release.

2) The simultaneous flattening of the waveforms on all three axes immediately following release is a direct indication that the ball is in free-fall during the loft phase. During loft, the ADXL waveforms are confined to the centripetal acceleration generated by the rotation of the ball.

3) The tilt sensing aspect of the accelerometer is apparent for as long as the ball remains in continuous contact with the lane, and is a direct indication that the ball is rolling.

4) The ball's impact with the pins is readily apparent from multiple closely spaced spikes in the waveforms, accompanied by a sudden increase in the noise content of those waveforms.

5) The extended free-fall period that follows impact with the pins is a reliable indicator that the ball has reached the end of the lane and is falling into the pit.

6) The content of a valid waveform, as described by the characteristics above, differs greatly from the content of the false activation examples presented in the paper. Although some of those characteristics may appear in the morphology of a false activation waveform, their order, duration, and amplitude appear to be unique to a valid waveform.

The primary caveat to all of the above is that the waveforms collected and analyzed so far originated from a single bowler, and are characteristic of that bowler's distinctive style. The waveforms that result from other bowlers with different bowling styles may differ greatly from the ones presented in this paper. Further data collection must be conducted across a wide variety of bowlers and bowling styles in order to develop truly generalized and robust autonomous operation of the *SenseModule*.

Now that it is possible to collect the 3-axis accelerometer data from within the bowling ball, the next phase of the project is to analyze the resulting waveforms in order to extract information useful to the bowler. However, that which is "visually obvious" to a human does not necessarily translate into a task that is easy and/or straight-forward to automate for a computer. The next section of this paper summarizes the author's work in identifying and developing automated algorithms that segment, filter, and analyze the raw data waveforms, and then extract useful bowling metrics from those results.

Section V: Waveform Deconstruction, Filtering, and Analysis

As presented earlier, the *REVMETRIX* system consists of three components: the *SenseModule* (autonomous data collection), the *ComModule* (data transfer), and the *RevMetrixApp* (data archival, analysis, and presentation). The *SenseModule*'s autonomous capability arises from the inclusion of algorithms for detecting activation, release, and shutdown that were developed iteratively through collection and analysis of the *SenseModule* raw data waveforms.

Apart from those automatic detection routines, the *SenseModule* makes no other decisions, and draws no other conclusions about the raw data. The *SM* does not need to know the ball's RPMs (angular velocity), nor how fast the ball is traveling down the lane (linear velocity) in order to perform its function. Rather, it identifies characteristics within the raw data in real-time that are indicative of the bowler delivering and releasing the ball, the ball rolling down the lane, impacting the pins, and then falling into the pit.

It is the *RevMetrixApp* (running on a smart phone, tablet, or PC) that will eventually receive the *SenseModule* data through the *ComModule*, and analyze and present the results to the bowler. This section of the paper presents preliminary analysis of the *SenseModule* raw data, and proposes methods for extracting useful bowling metrics from that data. Typical metrics of interest are release and impact speed and RPMs, axis tilt, loft distance, and delivery and release characteristics. This work is a precursor to the development of the *RevMetrixApp*.

Fast Fourier Transforms (FFTs), Finite Impulse Response (FIR) filters, and Wavelet decomposition and reconstruction are used to identify the distinct temporal elements of the 3-axis accelerometer waveforms that the *SenseModule* captures. After segmenting the waveforms, FIR and wavelet-based filtering techniques tuned to the morphology and frequency content of each distinct segment are used to analyze the segments and extract meaningful metrics for the entire waveform.

The raw sensor data is uploaded to a PC and stored in an Excel spreadsheet for off-line processing. MATLAB M-files import the raw accelerometer data and then employ a combination of FFTs and first-level *Haar* wavelet details from the 3-axis waveforms to identify four distinct phases (segments) in the temporal evolution of the waveform. After the waveform is segmented, both FIR filters and *biorthogonal* decomposition/reconstruction are used to extract the third-level approximation from the reaction region. Further filtering of the reaction region reveals the sinusoidal "chirp" signal indicative of the ball "revving up" as it approaches the pins. Extrapolation techniques are used to obtain meaningful data at the fringes of the segment.

For the purposes of this project, a Dell XPS i7-2670QM 2.20 GHz laptop with 8 GB of RAM, running Windows 7 (64-bit), served as the analysis platform. MATLAB student versions 2010a and 2012a, in combination with the MATLAB Signal Processing and Wavelet Toolboxes, were used to create and implement the analysis algorithms. Microsoft Excel 2010 was also used as a data analysis and presentation tool.

5.1 Acceleration Components

The *SenseModule* measures acceleration in three orthogonal axes (*X*, *Y*, and *Z*) referenced to the ADXL345 accelerometer. The *SM* is positioned in the finger hole such that the ADXL345's *X*-axis aligns parallel to the proximal axis of rotation (across the finger holes), the ADXL345's *Y*-axis aligns with the proximal direction of rotation (across the thumb hole), and the ADXL345's *Z*-axis passes through the proximal center of the ball. In Figure 25 below, the positive *X*-axis and *Y*-axis acceleration directions are shown, with the positive *Z*-axis acceleration pointing out of the page.

Note that the 3-axis acceleration origin is established by the *SenseModule*'s position within the ball, and does not have an external coordinate basis, such as the bowling lane. Although the ball travels in essentially the same direction every time – down a lane that is 60 feet long by 41.5 inches wide – the ADXL345's *X* and *Y* axes likely will not align with either of the lane's dimensions. In addition, the origin rotates with the ball, thus from the lane's point of view, the frame of reference is constantly changing.

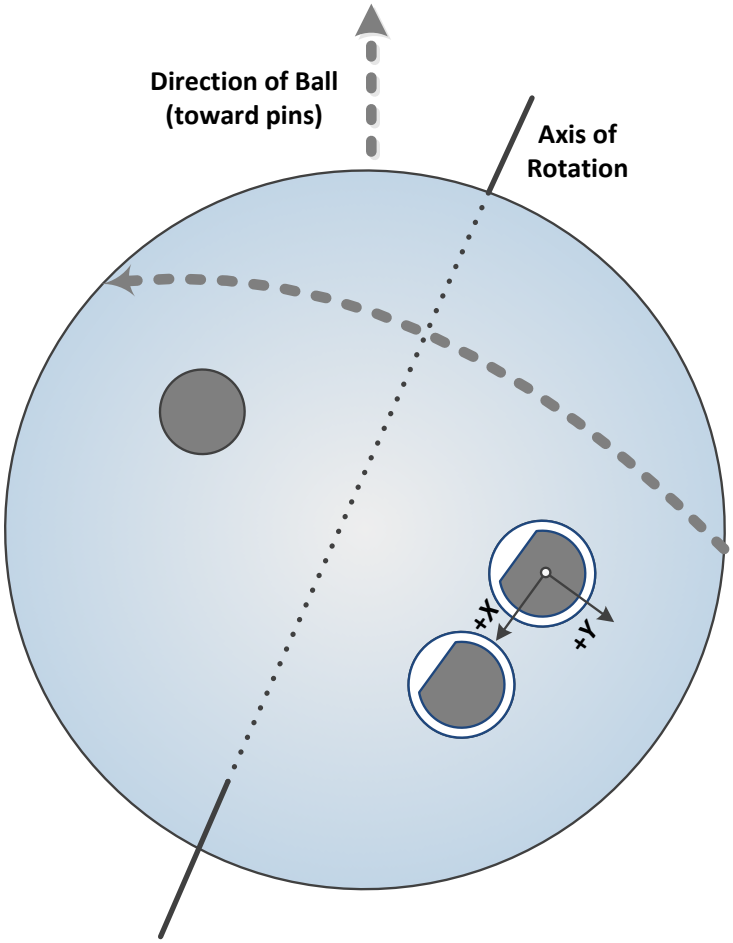

Figure 25: *SenseModule* Axis Orientation

The bowling ball can move through 6 degrees of freedom: linear motion in the *X*, *Y*, and *Z* directions, combined with angular motion in the *X*-*Y*, *X*-*Z*, and *Y*-*Z* planes. The ball also moves within Earth's gravitational field. The ADXL345 detects the ball's motion through those 6 degrees of freedom and aggregates three types of acceleration into the output for each axis:

1) **Linear:** Straight line acceleration.
2) **Angular:** Centripetal acceleration due to rotation.
3) **Gravitational:** Orientation with respect to gravity (tilt sensing, range of ± 1 *g*).

With the *SenseModule* oriented in the bowling ball as shown in Figure 25, the angular acceleration generated by the rotation of the ball registers as positive acceleration for the *X*-axis and negative acceleration for both the *Y* and *Z* axes.

For gravity (tilt sensing), whenever the ball is rolling on the lane, the tilt sensing aspect of the ADXL345 imposes a sinusoidal component on top of the angular acceleration component. The magnitude of the tilt component is limited to a range of ±1 *g*. The *Z*-axis reads +1 *g* when the finger hole containing the *SenseModule* points towards the ceiling and -1 *g* when the finger hole points toward the lane. The *Y*-axis reads +1 *g* when the finger hole is rotated 90° from the vertical in the direction of the thumb hole (ADXL *Y*-*Z* plane) and -1 *g* when rotated 90° away from the thumbhole. The *X*-axis reads ±1 *g* when the finger hole is rotated 90° from vertical in the plane of the finger holes (*X*-*Y* ADXL plane). See Figure 26.

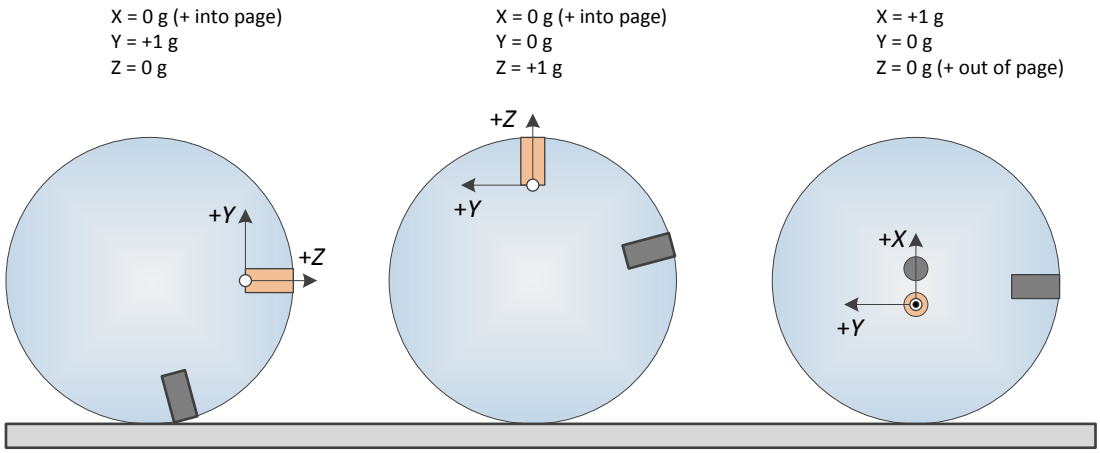

Figure 26: *SenseModule* Tilt Orientation

Linear acceleration is much more involved. Throughout the bowler's approach and delivery, the ADXL345 registers a combination of linear acceleration from the bowler's forward motion toward the foul line, and the ball's acceleration and tilt as it moves through the arc of the bowler's arm swing. It is not until the bowler begins to apply lift to the ball immediately before release that the ball starts to undergo rapid angular acceleration.

Whenever the ball is rolling on the lane, it experiences linear deceleration in opposition to its direction of travel due to the force of friction between the ball and the lane. The *SenseModule* rotates in the presence of that frictional force, which induces a similar effect as the tilt sensing aspect due to the

gravitational force, but orthogonal to the gravitational force. Thus, whenever the ball is in contact with the lane, the frictional force induces a sinusoidal element on the acceleration readings, as does the gravitational force. For any particular axis, those forces will be 90° out of phase, as the gravitational force is directed vertically downward, and the frictional force is directed horizontally backward, opposing the ball's linear motion.

When the ball is in free fall, e.g., during the loft phase after it has been released, and when it falls off the end of the lane into the pit, the ADXL345 only experiences the effects of angular acceleration. Those are the only times that the angular acceleration is completely isolated from the other accelerations.

The task at hand becomes to separate out the various types of acceleration from the raw data waveforms in order to recover the reaction of the ball, and then quantify that reaction for the bowler.

5.2 Raw Data Waveform Segments

The raw data waveforms captured for this paper exhibit a response that consistently evolves through various acceleration regions over the waveform sample time. Each waveform can be logically separated into the following segments by characterizing the amplitude and frequency content prevalent in the respective signals from those regions.

1) **Stance:** The period during which the bowler is relatively stationary, just before starting their approach. Steady-state (DC) acceleration due to gravity dominates at this time.
2) **Approach**: The time during which the bowler is delivering the ball to the lane. This segment is comprised of DC and low frequency data and concludes when the bowler begins to apply lift and turn to the ball immediately before releasing the ball.
3) **Release**: At the end of their approach, the bowler applies a sudden and forceful upward lift to the finger holes, while also turning the rotational axis of the ball counterclockwise (toward the left gutter) for right-handed bowlers.
4) **Loft:** This segment immediately follows the bowler's release of the ball. The bowler generally lofts the ball, which then travels several feet in the air before making its initial contact with the lane. It may then bounce one or more times before remaining in continuous contact with the lane. This segment is comprised of a flat DC-offset component due to angular acceleration, with several strong high frequency spikes (one for each impact with the lane) superimposed on that DC component. This segment concludes once the ball maintains contact with the lane.
5) **Reaction:** This segment comprises the time between the end of the LOFT segment and the ball's initial impact with the pins. The ball is rolling on the lane, and is interacting with the lane through the force due to friction between the ball and the lane. This segment is also characterized by a DC or low frequency offset component (angular acceleration), with a sinusoidal chirp response and high frequency noise superimposed on the waveform (tilt response combined with frictional opposition). The reaction segment ends when the ball impacts the pins.
6) **Pin Impact:** This segment starts at initial impact with the pins (generally the head pin), and continues until the ball starts to fall off the lane into the pit at the end of the lane. The pin impact segment is comprised of several high amplitude spikes (pin impacts), a good deal of high

frequency noise, along with a continuation of the sinusoidal tilt response, and a low-frequency offset bias due to the angular velocity of the ball.

7) **Shutdown:** The ball falls off the end of the lane and into the pit. The shutdown segment is characterized by the sudden transition of all three axes to steady-state acceleration (free fall). The *SenseModule* uses that DC signal to identify automatic shutdown.

5.3 Automated Segmentation

The segment boundaries of interest for analyzing the bowling ball dynamics after release are the release-loft boundary, the loft-reaction boundary, and the reaction-pin impact boundary. Each of those boundaries is demarcated by rapid changes in acceleration:

1) **Release-Loft:** A sudden increase in acceleration occurs as the bowler applies lift to the ball, followed by a sudden decrease in acceleration when the bowler releases the ball.
2) **Loft-Reaction:** Immediately following release, the ball is in free fall until it contacts the lane, generating one or more high amplitude impact spikes.
3) **Reaction-Pin Impact:** The ball impacts the pins, generating one or more high amplitude impact spikes.

The ambient light waveform is used first to find a common local starting point for the Release segment. The light signal increases fairly rapidly (from very close to 0) at the end of the release motion (see Figure 27). The portion of the light waveform following release is then used to recover the fundamental frequency of rotation F_R of the ball, which manifests itself from release through the first several revolutions of the ball. Alternatively, F_R can be determined by plugging the acceleration readings during loft and the *SenseModule*'s displacement from the center of the ball into the angular momentum equation. However, the technique implemented here does not require *a prioiri* knowledge of the depth of the *SenseModule* within the fingerhole.

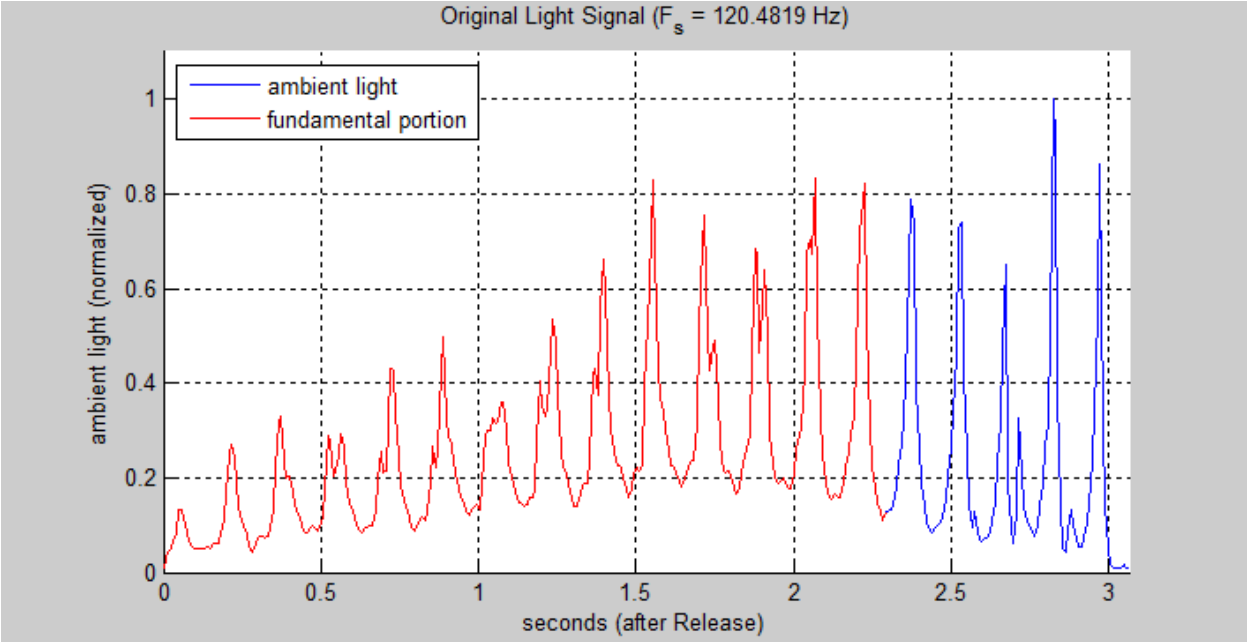

Figure 27: Ambient Light Waveform

After the time of release is identified from the light waveform, the remaining light signal is interpolated to 1 millisecond resolution (F_s = 1 kHz), and then a Hamming window is applied across that portion of the waveform. The result is then padded to yield a frequency resolution of 0.5 RPMs (0.00833 Hz) coming out of the FFT. F_R is then set to the greatest amplitude FFT frequency bin having a frequency greater than 60 RPMs (1.0 Hz). For example, from Figure 28, $F_R = V_a$, the peak angular velocity component (and fundamental frequency) of the ambient light waveform frequency spectrum. F_R is subsequently used for both wavelet decomposition and FIR filtering of the waveform.

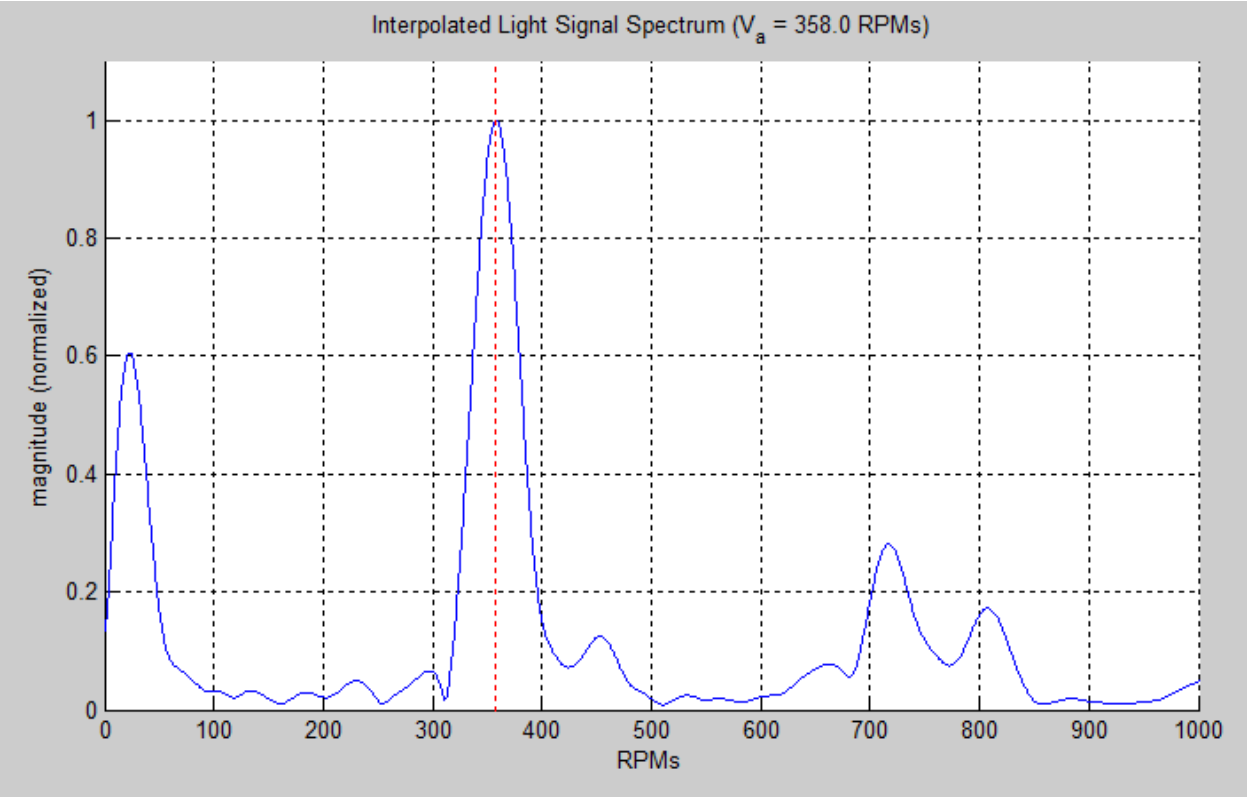

Figure 28: Interpolated Light Spectrum

The location of the above three segment boundaries can be found by applying techniques from Wavelet theory. Since Wavelet theory is based on repeated two-level decimation of the digital signal, the 3-axis accelerometer readings are first interpolated to yield 2^5 = 32 samples per Hz, based on the fundamental frequency F_R found above from the light waveform spectrum. That step causes each wavelet decimation level to occur at an integral harmonic of F_R, which yields cleaner results for this application.

Single-level *Haar* wavelet decomposition is used to obtain the high frequency details (impacts) of the 3-axis waveforms. The *Haar* wavelet was chosen because the acceleration readings contain both flat regions (free fall), and high amplitude spikes (impacts) that the shape of the *Haar* can easily detect. The details of the *Haar* wavelet decomposition are extracted, and Figure 29 shows the results of applying the *Haar* wavelet to the 3-axis acceleration signal. The red trace (***s***) is the vector magnitude waveform from the recombination of the three acceleration axes. The green trace ($\boldsymbol{d_1}$) shows the 1st-level *Haar* details extracted from the vector magnitude waveform. A similar process can also be applied to the individual acceleration axes.

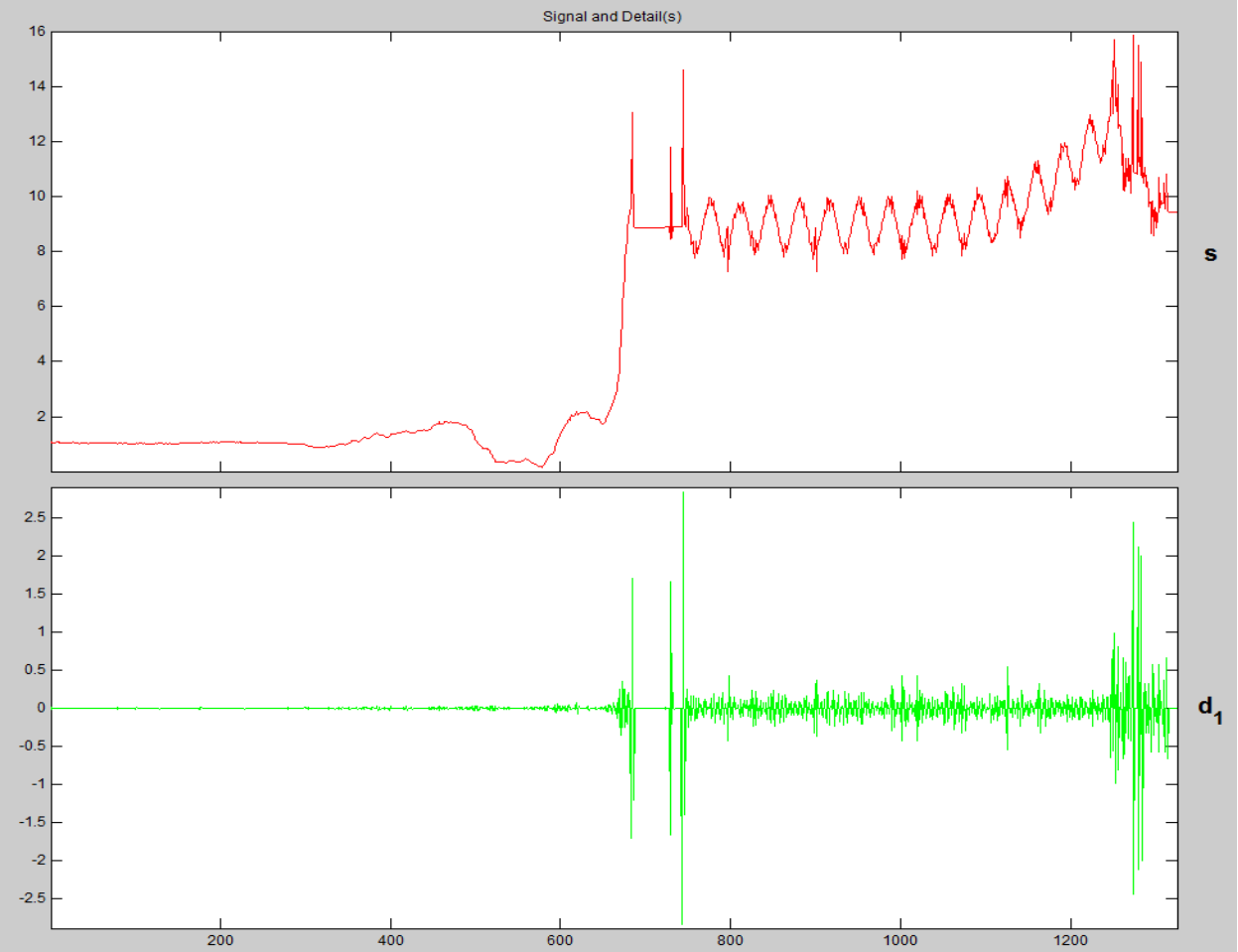

Figure 29: 1st-Level *Haar* Details (Impacts)

The vector magnitude details waveform is used to identify the significant high-frequency impacts in the original signals. Those impacts are used as starting points for locating the segment boundaries. Various statistical characteristics (mean, variance, standard deviation, 1st and 2nd derivatives) are then used to accurately localize the exact segment boundaries. Figure 30 shows the combined impact results, along with the 3-axis acceleration traces marked with the segmentation boundaries found from the automated segmentation algorithm using the impacts from d_1. The thresholds for identifying the significant impacts are shown as dotted lines.

Figure 31 shows the four segments that resulted from those boundaries. The APPROACH segment contains the stance, approach, and release regions that were described earlier. The IMPACT segment contains both the pin impact and shutdown regions. For the remainder of the paper, we will focus on the LOFT and REACTION segments. Our intent will be to isolate the linear deceleration, angular acceleration, the loft impacts, and the tilt response from each other, and then analyze and extract the useful bowling metrics from the resulting filtered component waveforms.

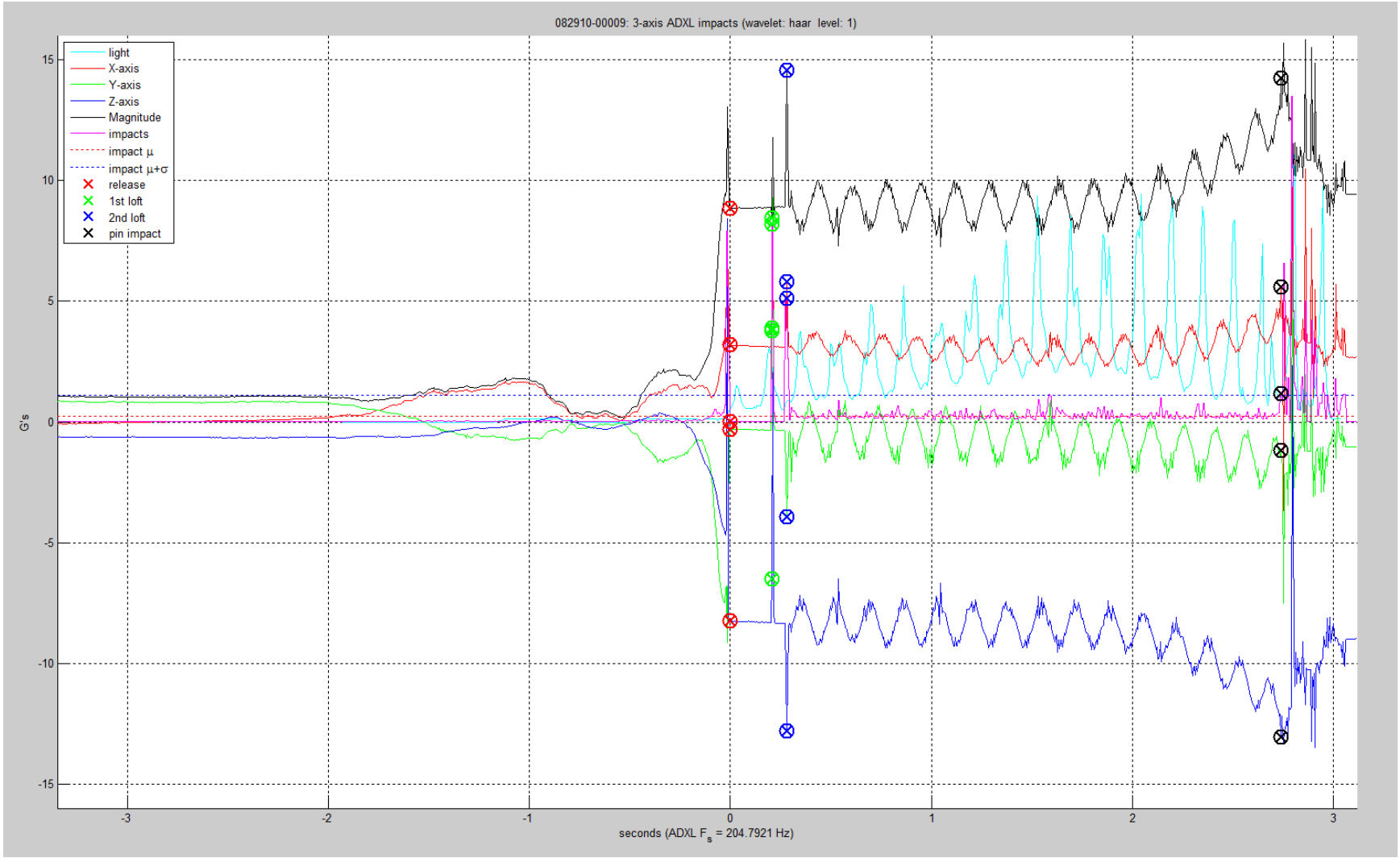

Figure 30: ADXL 3-Axis Segment Boundaries (Impacts)

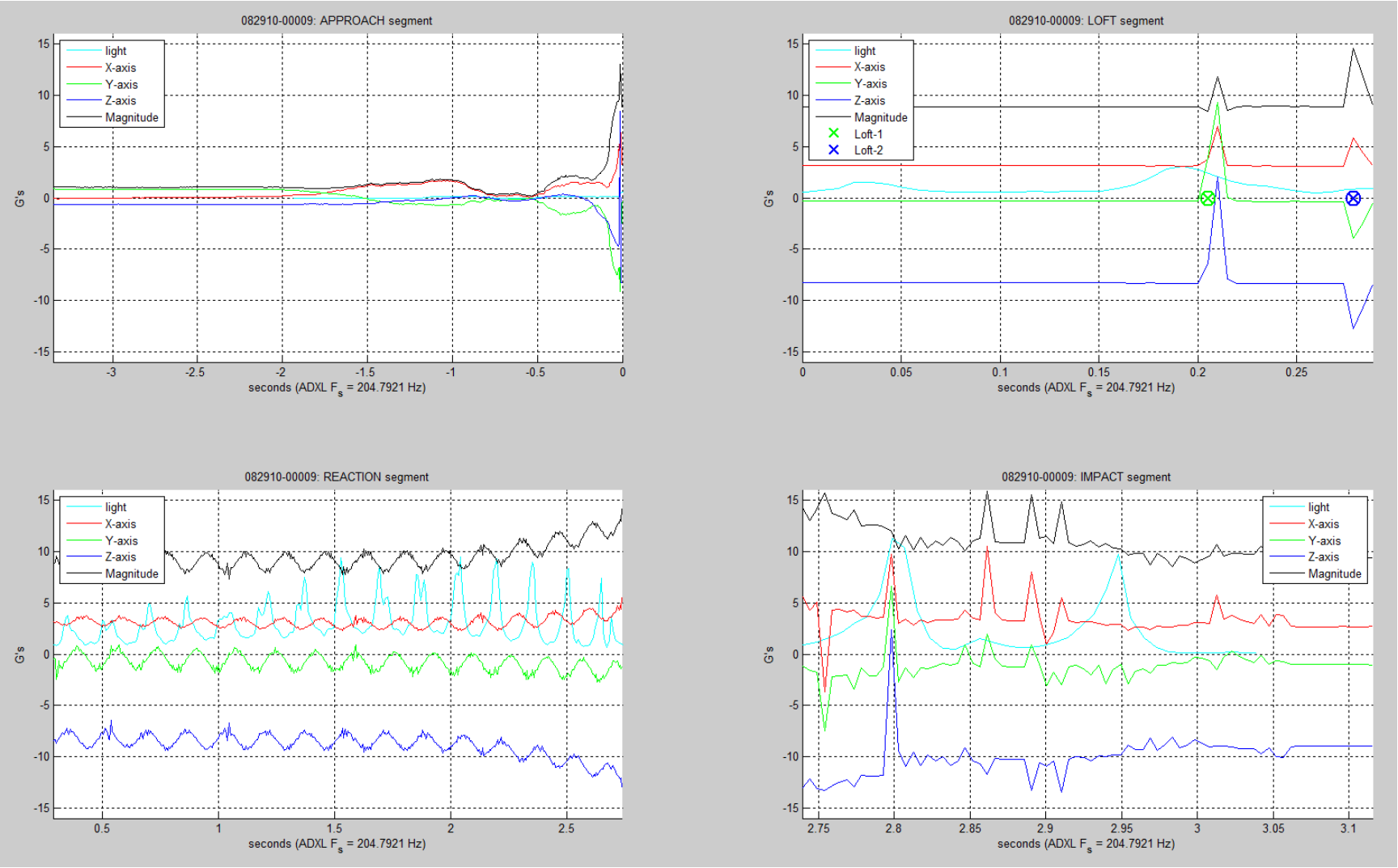

Figure 31: Raw Data Segments

5.4 Automated Waveform Deconstruction

The REACTION segment from Figure 31 is expanded below in Figure 32. The 3-axis acceleration waveforms of the REACTION segment are comprised of three distinct components:

1) **Angular acceleration:** Generated by the centripetal force due to the ball's rapid rotation.
2) **Sinusoidal tilt response:** From the *SenseModule* rotating through the gravitational field.
3) **High frequency noise:** From irregularities in the contact surfaces between the ball and the lane, *SenseModule* vibration in the finger hole, and digital noise infiltrating the ADXL345.

As the ball rolls down the lane, the force of friction between the ball and the lane converts the ball's translational kinetic energy to rotational kinetic energy, slowing the ball down, while increasing its angular velocity. The increase in angular velocity is visually apparent in the angular acceleration portion of the 3-axis signal of Figure 32, starting at 2.0 seconds, especially in the *Z*-axis signal.

As the angular velocity increases, the ball rotates more rapidly within the gravitational field, decreasing the period of the tilt response, resulting in a sinusoidal chirp signal. That chirp signal will become more apparent after isolating it from the composite signal, and filtering out the high frequency noise.

The amplitude of the angular acceleration and the frequency of the tilt response are directly related. As we will see, the phase and amplitude of the 3-axis tilt signals are also related to each other, and can be used to reveal the changing position of the ball's rotational axis.

Before we can accurately analyze the components of the acceleration waveforms, the above three components must be isolated from each other, while filtering out the noise component.

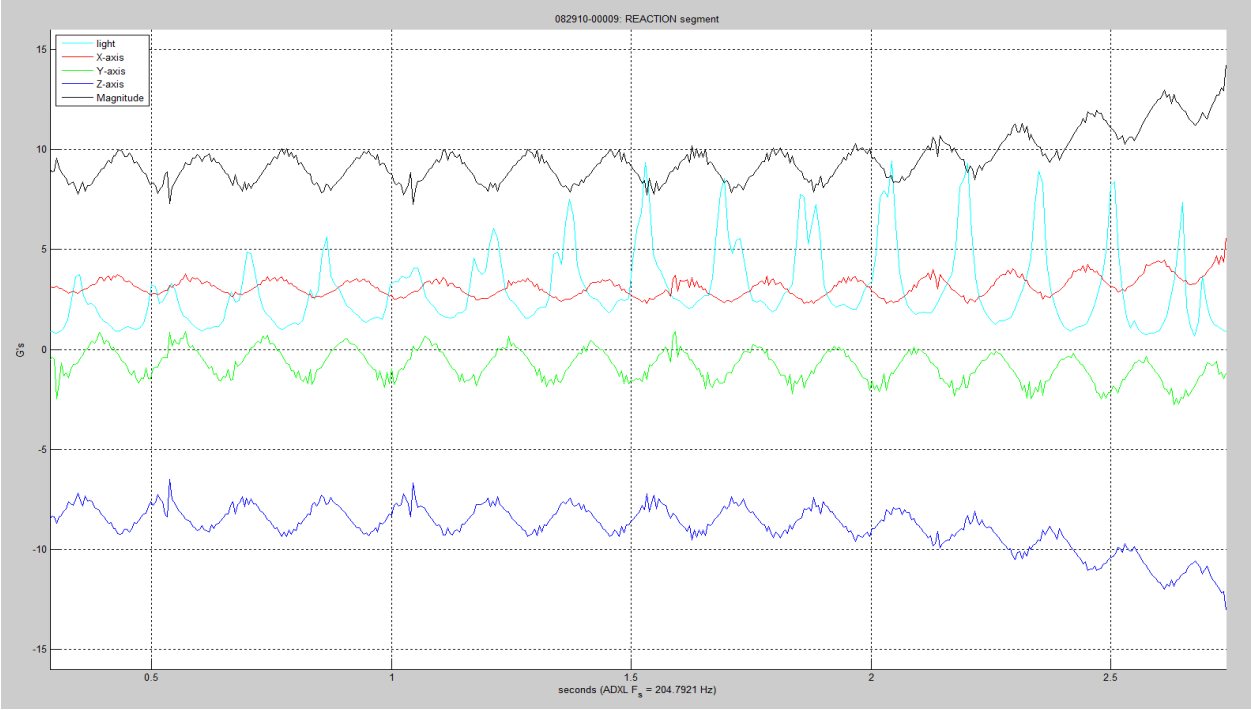

Figure 32: REACTION Segment

5.4.1 Waveform Deconstruction using Wavelet Decomposition

The high frequency noise portion of the acceleration waveforms has already been identified as the first-level component of the wavelet decomposition. We will use further decomposition to isolate additional components of those waveforms. The partial results of a five-level *biorthogonal* 6.8 wavelet decomposition and reconstruction of the *X*-axis acceleration waveform are shown in Figure 33 below.

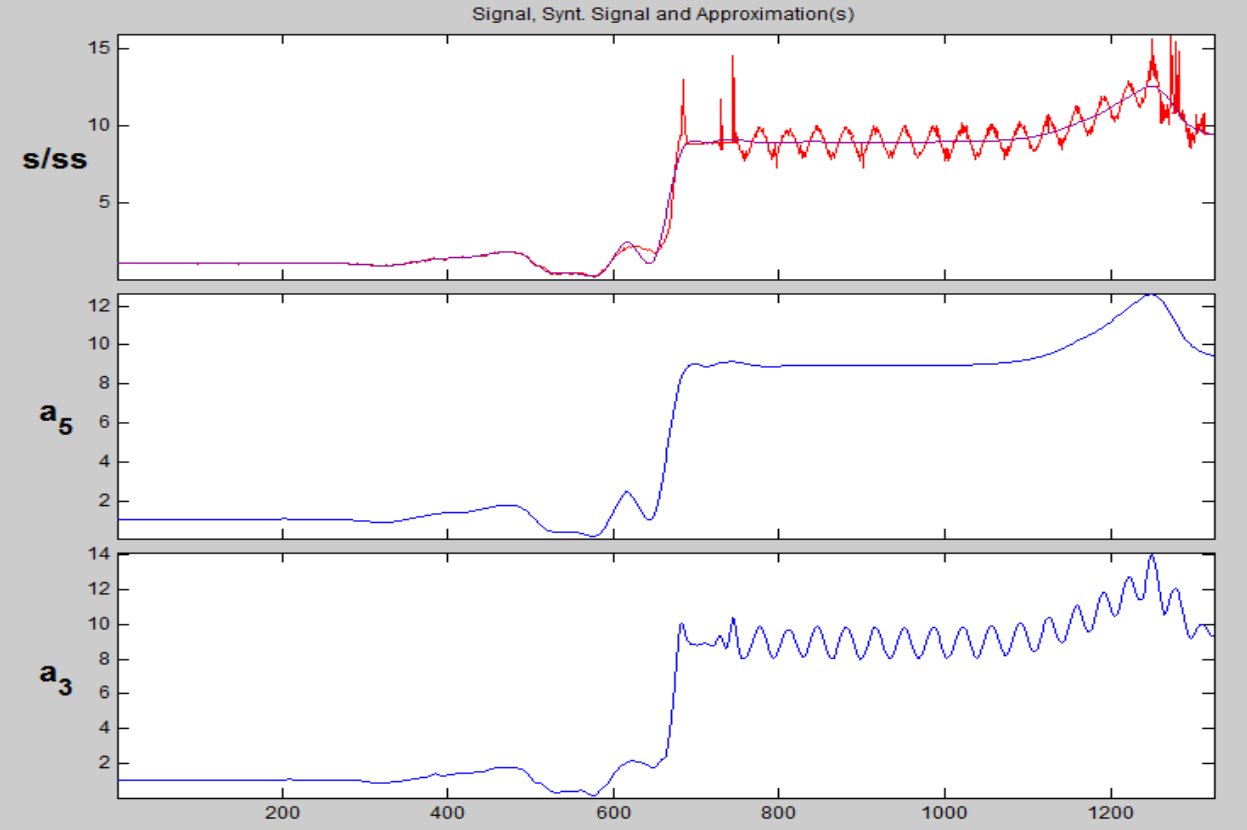

Figure 33: 3rd & 5th-Level *Bior6.8* Reconstruction

This is why we interpolated the original 3-axis waveforms into F_R x 2^n x T samples. Assuming that the high-frequency noise is far enough removed from the signal of interest, there will be some level of decimation (in this case levels d_1 to d_3) that filters out the noise and reveals the tilt response superimposed on the angular acceleration response ($\boldsymbol{a_3}$). At some further level of decimation (in this case levels d_1 to d_5), we see the isolated angular acceleration response ($\boldsymbol{a_5}$) emerge. The filtered angular acceleration response (***ss***) is superimposed on the original signal (***s***) to illustrate how well the $\boldsymbol{a_5}$ approximation conforms to the original signal. Subtracting $\boldsymbol{a_5}$ from $\boldsymbol{a_3}$ isolates the tilt response from being superimposed upon the angular acceleration, as seen in Figure 34 below. Wavelet decomposition/reconstruction works very well to separate out signal discontinuities, whereas an FIR filter is more appropriate for filtering noise out of cyclical waveforms. The acceleration signal, taken as a whole, exhibits both types of components. We have used wavelet decomposition to segment the waveform and isolate the components; we can now use an FIR filter on the cyclical portion of that waveform, which we have isolated from the REACTION segment.

5.4.2 Tilt Response

The results of the two different methods for isolating the tilt response are shown in Figure 34: wavelet decomposition/reconstruction (*wave*) and a conventional FIR filter (*FIR*). The *wave* signals are the a_5 – a_3 results from the previous section. The *FIR* signals are the result of filtering the REACTION segment using a Hamming window and applying a symmetric FIR filter to preserve the relative phases of the 3-axis waveforms. The fundamental frequency of rotation F_R was used as a basis to establish low and high cut-off frequencies for the pass-band FIR filter. In order to further attenuate the effects of discontinuity at the ends of the REACTION segment, the original signal was "folded" over at either end, before the Hamming window was applied, to extend the signal while preserving the frequency content.

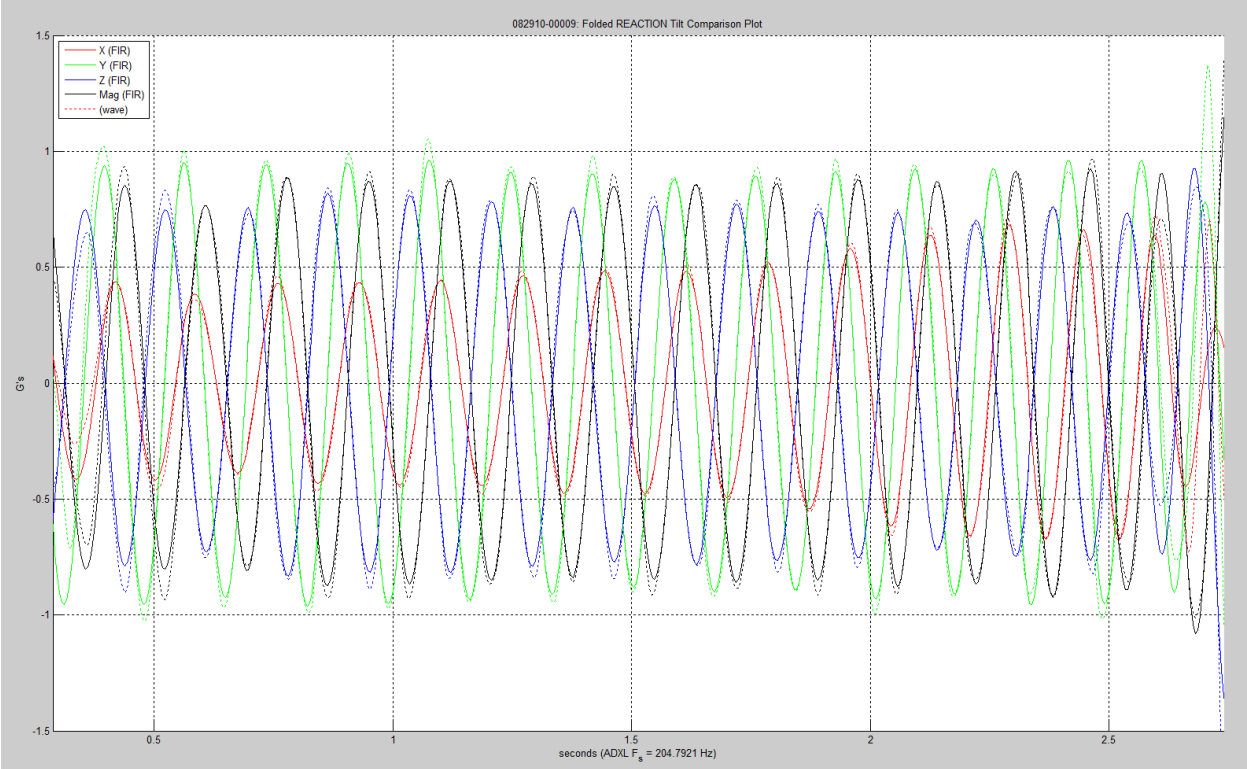

Figure 34: REACTION Segment Filtered Tilt Response

The wavelet-derived waveforms are shown with dotted lines, while the FIR-derived waveforms are shown with solid lines. Inspection of the wavelet and FIR results reveals a close correspondence between those signals, especially in the frequency and zero crossing regions. However, there is a small but discernible difference in the amplitudes, especially closer to either end of the REACTION segment, with the FIR versions displaying more uniform continuity than their respective wavelet versions. As a result, from this point forward, we will use the tilt responses that were obtained using the FIR filter.

Through a combination of wavelet decomposition/reconstruction and FIR filtering, we have now obtained clean sinusoidal waveforms from the tilt sensing aspect of the ADXL345 accelerometer. We will use the periods, amplitudes, and relative phases of those waveforms later to analyze the bowling ball's reaction as it rolled down the lane.

5.4.3 Angular Acceleration and Tilt Response

We have isolated the angular acceleration component and the tilt response components from each other. Figure 35 shows those two components, along with the raw data waveform (dotted lines) over laid on each other. The filtered tilt response closely follows the raw signal, and the angular acceleration component appears to be the running average of the tilt sensing component. This makes sense, as the sinusoidal tilt component was originally superimposed on the angular acceleration waveform.

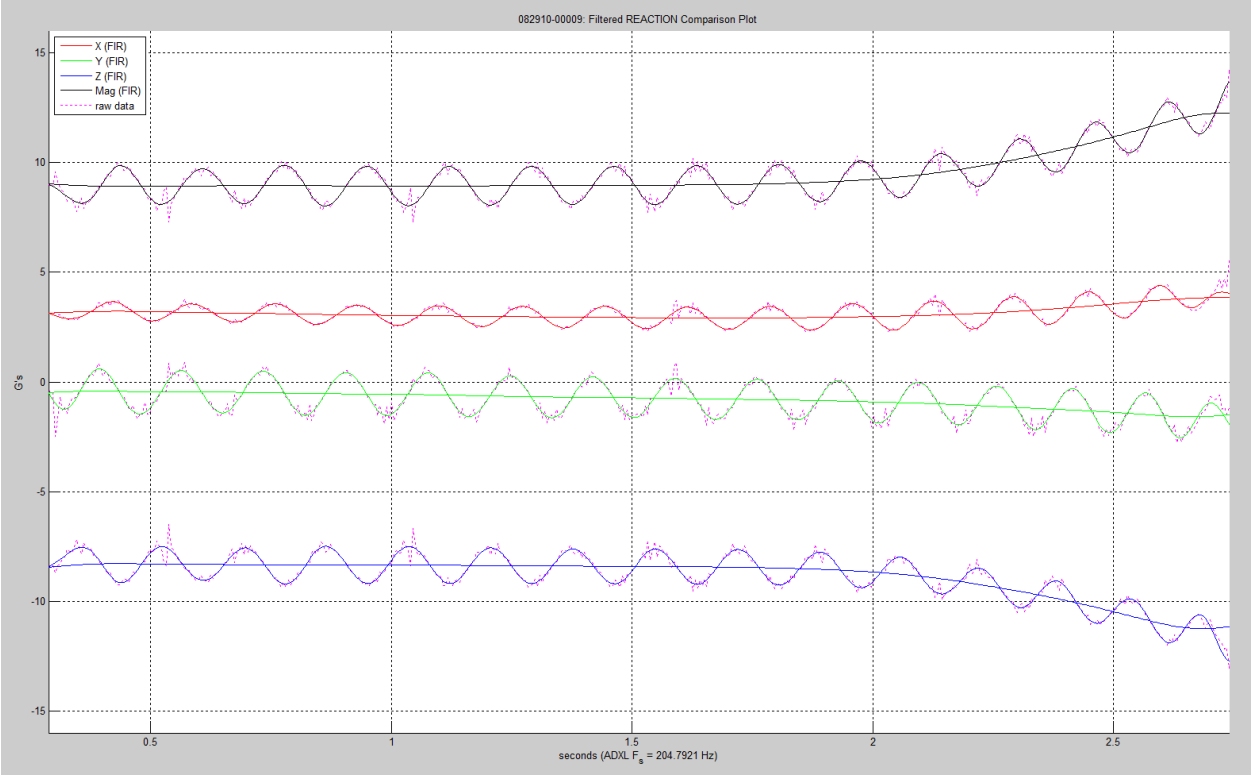

Figure 35: REACTION Angular Acceleration and Tilt Response

We will use the isolated angular acceleration waveforms in combination with the tilt sensing waveforms to reconstruct the instantaneous angular velocity of the ball, from release through impact with the pins.

5.4.4 Tilt Response Interpolation and Extrapolation

Having isolated the acceleration responses above, we can extrapolate the waveforms into the LOFT segment. Recall that the LOFT segment waveforms are flat, since the ball was in free fall, and that the only non-DC frequency content was due to the high amplitude impact spikes that were filtered out earlier along with the "noise" component. Figure 36 shows the tilt response extrapolated from the REACTION segment into the LOFT segment. Note that the graph now starts at 0 seconds (the release point), and spans the LOFT, REACTION, and PIN IMPACT segments.

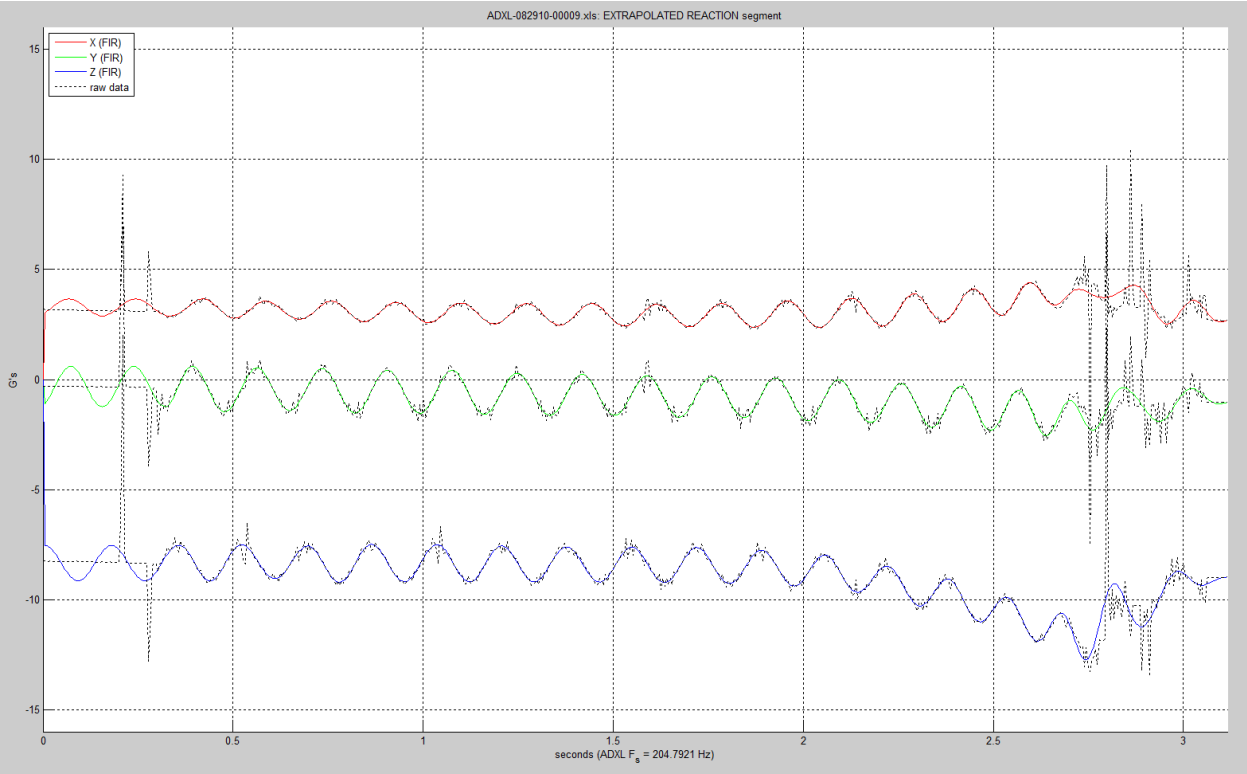

Figure 36: Extrapolated LOFT-REACTION Tilt Response

This graph is again shown with the filtered signals overlaid on the original raw data signals to show the high correspondence of the filtering techniques that have been used to isolate the various acceleration components from the original waveforms, as well as filter out the high-frequency noise.

5.5 Waveform Calculations

Now that we have isolated the various acceleration components, we can begin extracting the metrics relevant to the bowler. Metrics of interest related to the bowling ball that we can derive from the collected data are:

1) **RPMs:** Release, impact, and instantaneous angular velocity.
2) **Revolutions:** Revolution count from release through pin impact, revolution location.
3) **Ball Speed:** Release, impact, average, and instantaneous linear velocity.
4) **Loft:** Height and distance.
5) **Axis Tilt:** Release, impact, and instantaneous deviation of axis from parallel with lane surface.

Appendix D (page 124) includes an example of the typical output that the current MATLAB analysis program produces for one data set.

We can use the components of the waveform deconstruction effort just described to discover how the angular velocity changes over time. Having that knowledge, we can then use energy conservation techniques to find out how the linear velocity of the ball changes as the angular velocity changes. After we have determined the manner in which the linear velocity of the ball changes over time, it then becomes possible to determine the initial (release) velocity, which is one of the major execution variables that a bowler must learn to control. We can then locate the ball (along with each revolution) on the lane relative to the foul line, rather than relative to the time of release. That allows us to determine the ball loft distance (another major execution variable). Having found the conversion from time (sample index) to distance, we can then deduce the coefficient of friction acting between the ball and the lane, which can be used to infer the distribution of oil on the lane.

Since we know both the length of the lane and the time that it took for the ball to traverse the distance from the foul line to the pins, it is easy to calculate the average speed (linear velocity) of the ball. The average linear velocity provides us with a lower bound for the initial linear velocity the bowler applied to the ball at release, since the release velocity must have been greater than the average velocity (given that the linear velocity of the ball is always decreasing).

We utilized two different methods to find the angular velocity of the ball from the filtered waveforms. However, recovering the instantaneous linear velocity from the ball's average linear velocity and the angular velocity of each revolution presents an interesting, challenging, and non-trivial problem.

The following methods rely on the initial assumption that no energy is lost as friction works to transfer energy from the ball's linear kinetic energy to its angular kinetic energy. Those methods can be generalized to allow for a constant (but arbitrary) percentage of the energy transferred from the linear kinetic energy to be lost as heat/vibration due to friction.

The author's original paper [1] covered much of this ground, but from a "per-revolution" standpoint, rather than a "per-sample" standpoint, e.g., calculations for each revolution of the ball, rather than for each sample time. This portion of the paper provides updated derivations for the techniques presented in the original paper, along with the results of implementing those updated algorithms as part of this project.

Throughout this paper, we have assumed that friction is the only force of any significance acting on the ball. The results of that frictional force become apparent by observing the angular velocity of the ball increase as the force of kinetic friction acts to resolve the discrepancy between the initial linear and angular velocities of the ball.

Assume, for the moment, that the ball loses no energy throughout the course of a shot (the total kinetic energy of the ball remains constant). We can then draw the conclusion that all of the energy the ball gains from the increase in its angular velocity must have been transferred from the ball's linear velocity, i.e., any increase in angular kinetic energy must be exactly offset by a decrease in linear kinetic energy.

Based on our assumption of constant energy, the energy of the ball at any instant (sample time) must also be constant. Since the angular velocity for each revolution of the ball, and for each sample time, has already been found, if the total kinetic energy of the ball for a specific period is known, the linear velocity for that period can also be found. Once the linear velocity at any sample time is known, it is then possible to find the ball loft distance, and the location of each revolution of the ball, relative to the foul line.

Putting those assumptions and deductions into more formal terms, the ball possesses constant energy throughout its trip to the pins, and its energy at any instant is equal to its energy at any other instant.

The total energy of the ball (E) is the sum of its potential (P) and kinetic (K) energies,

$$E = P + K$$

Since the ball rolls on a flat, level lane surface, there is no potential energy ($P = 0$), thus

$$E = K$$

Since it is assumed that the ball has constant energy from its release at the foul line to its impact with the pins, then for K_R (energy at release), K_P (energy at pin impact), and K_i (energy at any sample point i),

$$K = K_R = K_i = K_P$$

The assumption of constant energy implies that energy losses due to axis torque, vibration, heat, air resistance, and noise generation are negligible. Therefore the only components of the energy of the ball are its angular kinetic energy (K_ω) and its linear kinetic energy (K_v), thus

$$K = K_\omega + K_v$$

The linear kinetic energy of an object with mass m and linear velocity v is given by

$$K_v = \frac{1}{2}mv^2$$

The angular kinetic energy of an object with mass m, moment of inertia I, and angular velocity ω is

$$K_\omega = \frac{1}{2}I\omega^2$$

The moment of inertia I of a sphere with mass m and radius r has the form

$$I = kmr^2$$

The value of *k* is determined from the mass distribution within the sphere. We can find *k* from the United States Bowling Congress (USBC) specification for the *Radius of Gyration* (*RoG*) of a bowling ball. The USBC imposes limits on *RoG* of 2.430" to 2.800" [10].

$$(RoG)^2 = \frac{I}{m}$$
$$= \frac{kr^2 m}{m} = kr^2$$
$$k = \left(\frac{RoG}{r}\right)^2$$

The USBC also imposes limits on the radius *r* of a bowling ball such that

$$4.250" \leq r \leq 4.295"$$

Combining the USBC limits for *RoG* and the radius of a bowling ball yields

$$\left(\frac{2.430}{4.295}\right)^2 \leq k \leq \left(\frac{2.800}{4.250}\right)^2$$
$$0.3201 \leq k \leq 0.4340$$

Thus, the range for the moment of inertia for a bowling ball of mass *m* is

$$(0.3201)\left(\frac{4.250}{12}\right)^2 m \leq I \leq (0.4340)\left(\frac{4.295}{12}\right)^2 m$$
$$0.0402m \leq I \leq 0.0556m, \qquad (\text{lb} - \text{ft}^2)$$

The USBC requires bowling ball manufacturers to measure the *RoG* for each model of bowling ball they sell, and that value is available to use in the calculations that follow.

Substituting the moment of inertia into the equation for angular kinetic energy, we get

$$K_\omega = \frac{1}{2} I\omega^2 = \frac{1}{2} kmr^2\omega^2$$

Therefore, the total kinetic energy *K* of the ball is given by

$$K = \frac{1}{2}mv^2 + \frac{1}{2}kmr^2\omega^2$$
$$= \frac{m}{2}(v^2 + kr^2\omega^2)$$

Having established the energy equations and the range for moment of inertia, and value for *k*, we can now move on to establishing the calculations for the bowling ball metrics using the waveform components we previously extracted.

5.5.1 Average Ball Speed

The average ball speed (linear velocity) can easily be found from the length of lane (60 feet) and the time elapsed from release of the ball to its initial impact with the pins. If we let *D* be the distance from the foul line to the head pin, and T_s be the elapsed sample time from release to pin impact, then

$$v_{ave} = \frac{D}{T_s} = \frac{60}{T_s}$$

The above equation assumes that the ball is released at the center of the foul line, travels in a straight line, and makes contact with the center of the head pin at a distance of 60 feet from the foul line. In reality, none of those assumptions is precisely true. A discussion on the errors introduced by the assumptions made in the paper is presented in Section 5.6.

5.5.2 Revolution Period and Count

It is a straight-forward process to locate and count the peaks and valleys in the tilt waveforms. Figure 37 depicts the temporal locations of the half-revolutions in the REACTION segment tilt response. We previously extrapolated the tilt response from the beginning of the REACTION segment into the LOFT segment. We can extrapolate the partial revolutions immediately before pin impact from the periods and rates of change of the half-revolutions at the end of the REACTION segment.

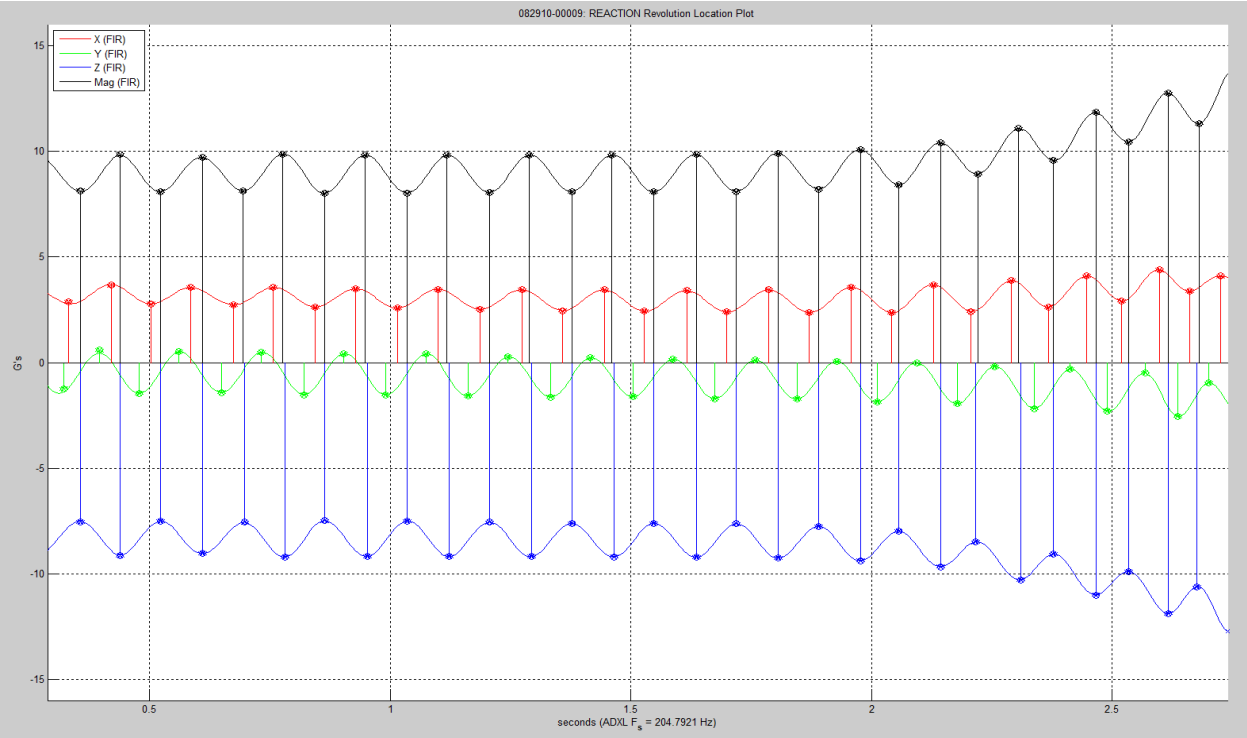

Figure 37: REACTION Revolution Location

The half-periods (peak-to-valley and valley-to-peak) of the 3-axis waveforms are counted and then averaged to come up with the revolution count. The difference in sample times between the peaks and valleys are then used to determine the period of each half-revolution. We will use that information next to determine the instantaneous angular velocity of the ball.

5.5.3 Instantaneous Angular Velocity (RPMs)

Having isolated the tilt response, we can now determine the peak-to-valley and valley-to-peak times for each half-revolution for each tilt waveform from Figure 34. The half-revolution angular velocities (in RPMs) for the revolution containing peak *p* and valley *v* are given by

$$f_{(v-1,p)} = \frac{2}{t_p - t_{v-1}}(60) \quad (\text{valley} - \text{to} - \text{peak})$$

$$f_{(p,v)} = \frac{2}{t_v - t_p}(60) \quad (\text{peak} - \text{to} - \text{valley})$$

We can extract the angular velocity for the LOFT segment from the average angular velocity of the first 250ms to 500 ms of the results in the REACTION segment, since there is generally little friction in that part of the lane to generate changes in angular velocity. Figure 38 below shows a plot of the results.

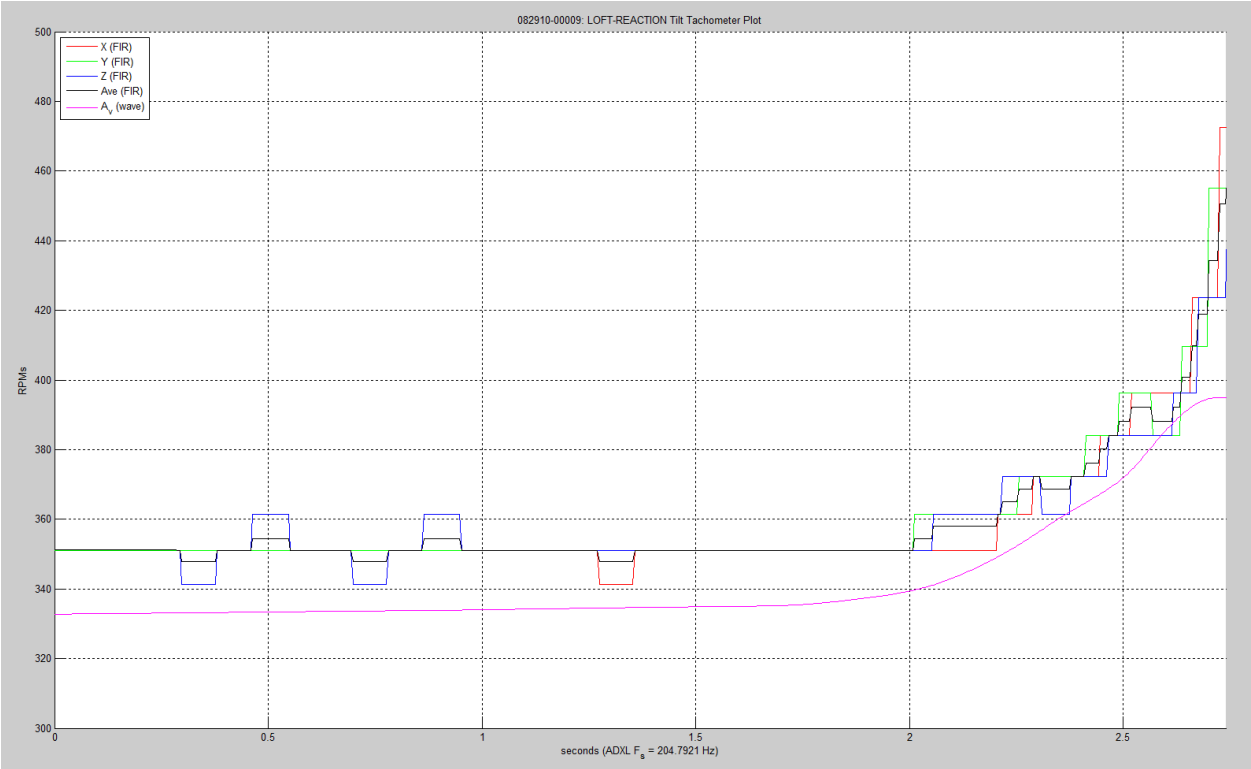

Figure 38: LOFT-REACTION Angular Velocity

Notice the effects of the granularity of our measurements. We previously interpolated the waveforms to have 32 samples per Hz of F_R, which results in a resolution of ~10 RPMs. Thus a jitter of one sample time in a full-revolution period translates to a step change of 10 RPMs, and a corresponding step change of 5 RPMs in the half-period results. Such step responses will cancel each other out over multiple periods, as indicated by the average waveform. We can apply a fifth-order polynomial curve-smoothing routine to the average waveform to reconstruct a smooth instantaneous angular velocity curve from the tilt response. We could also use a higher level of interpolation to increase the RPM resolution: $2^8 = 256$ samples per Hz of F_R would yield a resolution of ~1 RPM.

Figure 38 above also displays the angular velocity curve that we obtained directly from the low frequency portion of the magnitude acceleration waveform. That acceleration is largely due to the centripetal acceleration generated by the rotation of the ball.

We can extract the angular velocity *f* from the centripetal acceleration A_c, as follows:

$$A_c = \frac{v^2}{r} = \frac{\omega r^2}{r} = \omega^2 r, \qquad \text{where } v = \omega r = 2\pi f r.$$

$$\omega = \sqrt{\frac{A_c}{r}}$$

The revolution rate *f* in RPMs is then

$$f = \frac{\omega}{2\pi}(60) \text{ RPMs}$$

During the LOFT segment, it directly indicates the centripetal acceleration, since the ball is in free fall during that time. Figure 38 reveals an approximately 5% discrepancy between the angular velocity we extracted from the tilt response and that obtained from the angular velocity curve. Ideally those two curves should more closely match, and the author has yet to adequately account for the discrepancy.

5.5.4 Instantaneous Linear Velocity (Ball Speed)

Keeping in mind that any increase in angular velocity produces a corresponding decrease in linear velocity, we can now determine the deceleration of the ball from the change in instantaneous angular velocity. That determination then allows us to develop an equation that produces the linear velocity v_i for each sample period *i* of the waveform.

Recall that distance is the integral of velocity with respect to time. In the case of a sampled system, that relationship is given by the summation

$$D = \sum_{i=0}^{n-1} v_i t_s, \qquad \text{where } t_s \text{ is the sample period } \left(t_s = \frac{1}{f_s}\right).$$

Recall the assumption that the kinetic energy *K* of the ball remains constant from release to pin impact,

$$K_0 = K_i = K_n, \qquad \text{for } 0 \le i \le n$$

If we let K_i be the kinetic energy of the ball during sample time *i*, then

$$K_i = \frac{m}{2}\left(v_i^2 + k\omega_i^2\right)$$

Combining the above two equations yields

$$\frac{m}{2}(v_0^2 + k\omega_0^2) = \frac{m}{2}\left(v_i^2 + k\omega_i^2\right)$$

Solving for v_i, we get

$$v_i = \sqrt{\left(v_0^2 + k\left(\omega_0^2 - \omega_i^2\right)\right)}$$

For constant energy, the above equation assumes that friction acts solely to transfer energy from linear kinetic energy to angular kinetic energy. We can now obtain an expression for each linear velocity v_i in terms of the initial linear velocity v_0. Substituting into the summation yields

$$D = \sum_{i=1}^{n} \sqrt{(v_0^2 + k(\omega_0^2 - \omega_i^2))} \cdot t_s$$

We can now develop a converging iterative solution for v_0, and then generate the remaining v_i values from v_0. To start the iteration, we need an initial "seed" value for v_0. We know that D = 60 feet, and an appropriate first guess is the average linear velocity v_{ave}, which we found earlier, thus

$$v_0 = v_{ave} = \frac{D}{T_s} = \frac{60}{T_s}$$

The final value for v_0 must be greater than v_{ave}, since the ball slows down after release. Evaluating the summation with v_{ave} results in a value D' < 60. We can then use D' to arrive at the next guess, as follows

$$v_0' = v_0 + \frac{(60 - D')}{T}$$

The term (60 - D')/T represents the error in the average linear velocity distributed across each sample point. The adjustment to v_0 adds/subtracts that discrepancy to create the next value for v_0. Iteration continues in this fashion (guess, calculate, adjust), until the difference 60 – D' falls within an acceptable error margin, at which point we have found the true initial linear velocity v_0. We can then find the linear velocities for each sample point i by plugging the values for v_0, ω_0, and ω_i into

$$v_i = \sqrt{v_0^2 + k(\omega_0^2 - \omega_i^2)}$$

Figure 39 shows a plot of the change in linear velocity with respect to time, from the above calculations.

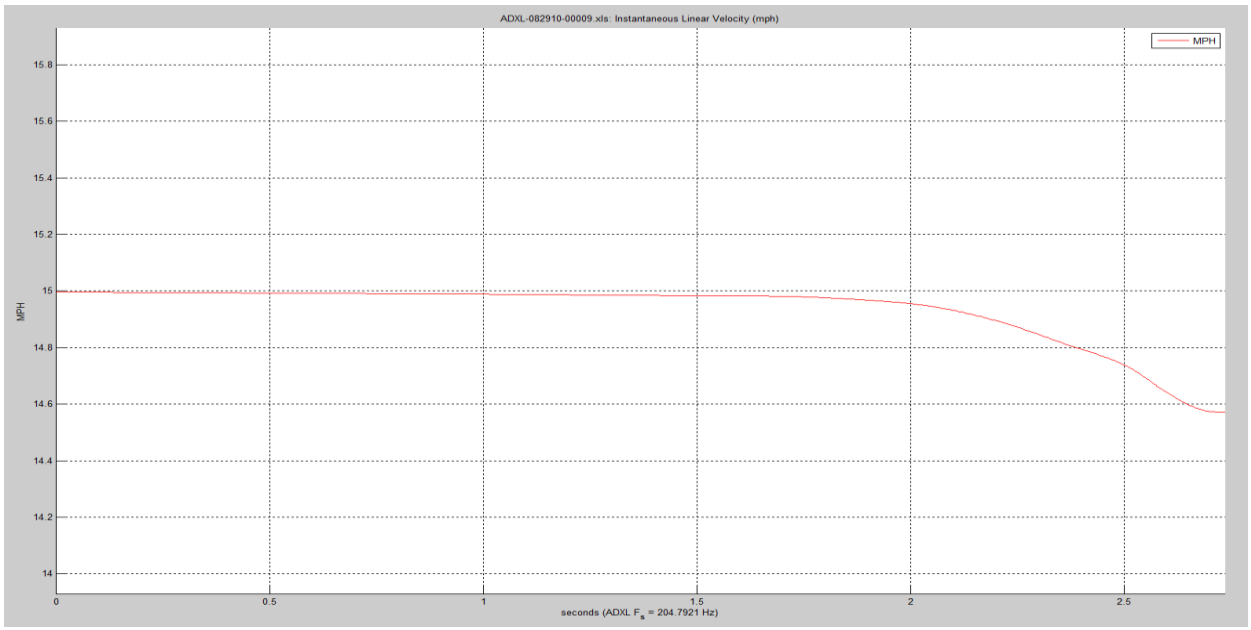

Figure 39: Instantaneous Linear Velocity

5.5.5 Distance

We can now use the instantaneous linear velocity to find the distance the ball covered during each sample period. We can then calculate the location (relative to the foul line) of every point between release and pin impact, enabling us find the distance the ball was lofted, locate each revolution relative to the foul line, and relate the linear and angular velocities of the ball with respect to distance.

The distance D_k that the ball has travelled at any time point k since release at the foul line is given by

$$D_k = \sum_{i=0}^{k-1} v_i t_s, \qquad \text{where } t_s \text{ is the fixed sample period } \frac{1}{f_s}.$$

Now that we can cross-reference sample points with lane distance, we can easily find the loft distance, as well as the location of each revolution on the lane. Those revolution locations (the angular velocity relative to lane distance) reveal where the ball begins to experience significant angular acceleration (the angular velocity begins to increase). That distance is called the *break point* of the ball, and the *break point* moves as the lane oil distribution changes during the course of a bowling session.

5.5.6 Loft Height and Distance

Since we previously found the time stamps for the loft impacts, it is an easy matter to now get the distance the ball flew in the air past the foul line before it first hit the lane. If t_R is the time of release and t_{L1} is the time of the first loft impact, we can also calculate the loft height from the "time of flight" of the ball after release using the equation for projectile height,

$$h = \frac{1}{2} g (t_{L1} - t_R)^2, \qquad \text{where } g \text{ is the acceleration due to gravity.}$$

There will be some uncertainty here, as we will not be able discern how high above the lane the ball was released, without some additional analysis of the release waveform. But this should at least provide a good first approximation.

5.5.7 Coefficient of Friction

A bowling ball generally skids the entire length of the lane, and thus it experiences sliding kinetic friction between the ball and the lane the entire time. We can recover the coefficient of kinetic friction from the previous calculations. Under the assumptions, the force due to kinetic friction is the only force acting between the ball and the lane, and that force acts solely to transfer linear kinetic energy to angular kinetic energy. Since the changes in angular and linear velocity for each sample are known, we can find the frictional force required to generate those changes.

Since we're assuming that the kinetic frictional force is the only input of any consequence to the system, we can rewrite the standard equation for work as

$$W = F_\mu \cdot d$$

Solving for the force due to friction, we get

$$F_\mu = \frac{W}{d} = \frac{\Delta K}{\Delta d}$$

We need the change in kinetic energy that the frictional force generated, over the distance the ball travelled while that force was being applied. For any sample point *i*, the frictional force $F_{\mu i}$ acting during *i* is given by

$$F_{\mu i} = \frac{\Delta K_{\omega i}}{\Delta d_i} = \frac{km}{2}\left(\frac{\omega_i^2 - \omega_{i-1}^2}{d_i - d_{i-1}}\right) = -\frac{m}{2}\left(\frac{v_i^2 - v_{i-1}^2}{d_i - d_{i-1}}\right)$$

Note that we can use either the change in angular kinetic energy, or the change in linear kinetic energy to find the kinetic frictional force. We can then obtain the coefficient of kinetic friction μ_i between the ball and the lane for any sample point *i* from

$$\mu_i = \frac{F_{\mu i}}{mg}, \qquad \text{where } g \text{ is the acceleration due to gravity}$$

Combining the equations, we get

$$\mu_i = \frac{k}{2g}\left(\frac{\omega_i^2 - \omega_{i-1}^2}{d_i - d_{i-1}}\right) = -\frac{1}{2g}\left(\frac{v_i^2 - v_{i-1}^2}{d_i - d_{i-1}}\right)$$

Figure 40 shows a graph of the coefficient of kinetic friction relative to distance from the foul line. The graph reveals the limitations of our assumption that the ball experienced no loss of kinetic energy due to the other forces in the system. Although the values for μ are reasonable in the latter third of the graph, It is unlikely that μ is so close to zero for the first 35 feet of the lane - typical minimum values for μ are 0.05 – 0.1 in that area. Given that the ball is rotating rapidly while skidding down the lane, there is certainly a loss due to friction. Also, the sudden drop off in the last 4-5 feet is the result of the FIR filter causing the angular velocity curve to roll off in the region close to the IMPACT segment. Those limitations, along with how to address them, will be discussed in Section 5.6.

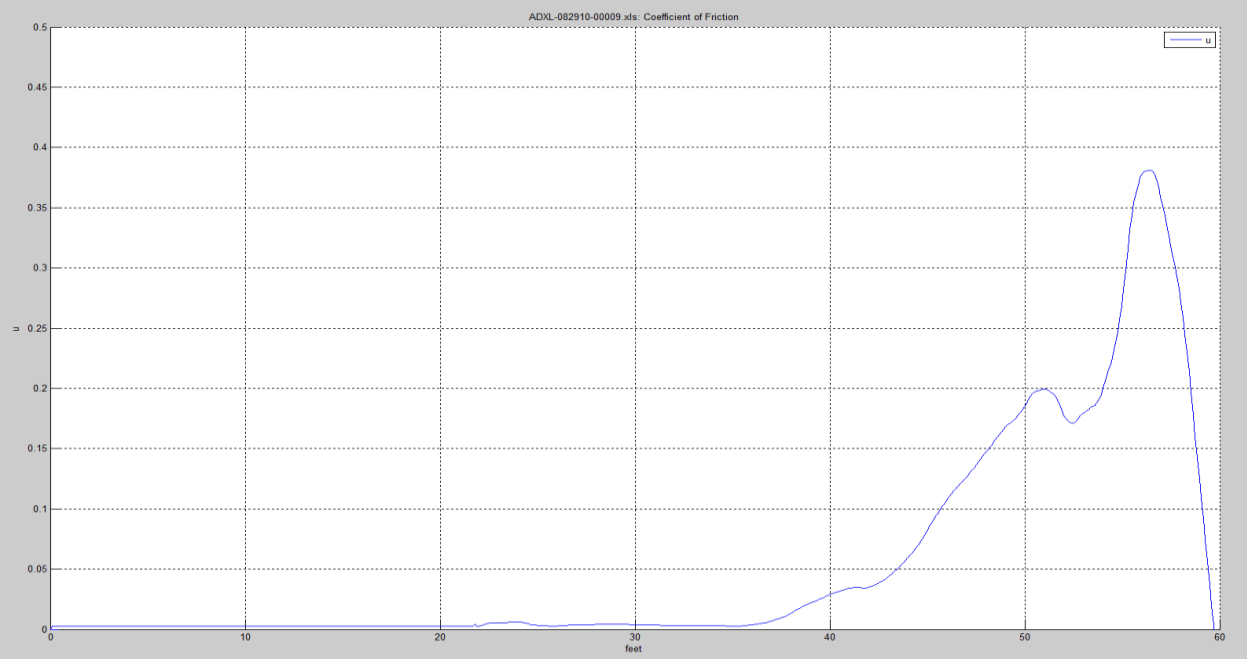

Figure 40: Coefficient of Friction

5.6 Assumptions and Error Analysis

Throughout the presentation of the raw data analysis, and the derivations of the bowling metric calculations, we have relied on some very basic assumptions. This section presents those analysis assumptions in greater detail, along with an assessment of their possible error contributions. The author presented a similar analysis and summary in his first paper, and the following analysis borrows from that previous discussion.

5.6.1 Distance

We have assumed that the ball travels 60 feet - the distance from the center of the foul line to the center of the head pin. The bowler normally releases the ball close to the foul line, and the first pin the ball encounters is the head pin (at least on the first ball of any frame). Let us take a closer look at what those assumptions really mean.

- **The ball is released at the foul line:** Generally, the bowler releases the ball at, or just beyond, the foul line. If the bowler releases the ball beyond the foul line, the distance the ball travels is shorter than 60 feet, and the calculated velocity will be less than the actual velocity. Since the bowler's feet must stay completely behind the foul line in order to legally deliver the ball, the distance they can reach beyond the foul line is limited to 12-18" at release of the ball.

  Conversely, if the bowler releases the ball behind the foul line, the distance the ball travels is greater than 60 feet, and the calculated velocity will be greater than the actual velocity. Releasing the ball behind the foul line is usually the result of a noticeable lapse in execution on the bowler's part (dropping the ball, releasing the ball early, or stopping short of the foul line).

  The nominal margin of error at the point of release is estimated to be 6" to 18" beyond the foul line, which introduces an error in the distance the ball travels of -12".

- **The ball hits the head pin at a fixed location:** At the other end of the lane, the ball can hit the headpin in different spots - head on, or on either side. A pin is 4.75" in diameter at the height at which the ball contacts it, and the center of the pin is located 60' ±½" from the foul line. A bowling ball is nominally 8.55" in diameter. Assuming the ball is released with its center over the foul line and it hits the head pin dead center, it must have travelled 59' 5" to 59' 6" from the point of release. If the ball barely grazes the right or left side of the head pin, then the ball traveled 59' 9" to 59' 10". A solid pocket hit (the goal of any potential strike delivery) falls halfway between those two ranges, so the expected distance from the foul line that the initial head pin impact occurs is 59' 8". Therefore, the error introduced in hitting the head pin is -4"

- **The ball takes the shortest path from the foul line to the head pin:** In reality, the ball is rolled with some amount of hook, and is released from some place other than the center of the lane, and is initially directed somewhat toward the right gutter (for a right-handed bowler). If the ball were thrown from the outside edge of the right gutter in a straight line 45-50 feet down the lane, and then suddenly hooked straight toward the pocket, it would travel an extra $^{7}/_{8}$". At the other extreme, if the ball were released at the left gutter, and traced a regular arc to the right gutter at 30 feet, and then back to the head pin, it would travel an extra 6". This is an extreme amount of hook, but will be factored into the error budget as ±3".

- **The ball and pins each have fixed diameters:** The ABC maintains tight control over the allowed dimensions of the ball and the pins. The ball has a nominal diameter of 8.545 ±0.045", resulting in a nominal circumference of 26.855 ±0.141". The diameter of the ball contributes a negligible error. At the height that the ball strikes a bowling pin, the pin must have a nominal diameter of 4.766 ±0.031". That value is also an insignificant contribution to the error calculations.

Combining the three significant errors (at the foul line, at the pins, and from the path of the ball), yields a total of approximately -13" to -19" of error in the distance the ball travels. For a nominal distance of 60 feet, the total error works out to be -1.81% to -2.64%. For a ball thrown at 15 mph, this error will manifest itself as an increase in the calculated average velocity from 15.27 mph to 15.40 mph.

In the final *RevMetrix* application, the bowler could input their normal release point as a configuration parameter. Then, for each ball thrown, they would indicate the approximate board where the ball was released, the extreme right (or left) board that the ball crossed on its path to the pins, and the board the ball was on at the time of impact. The *RevMetrix* application would then utilize this information in its velocity calculations. The margin of error in the distance the ball travelled would likely be reduced to less than 6", increasing the accuracy of the calculation to within 0.1 mph (an error of about 0.7%).

5.6.2 Time

Another assumption is that the microprocessor's clock is accurate. The smaRTClock is used to time stamp the raw data samples, and that clock is based on a 32.768 kHz watch crystal with an error over its operational temperature range of no more than ±200 ppm (0.02%), or 600 μs over a 3-second shot, which is an insignificant error contribution.

5.6.3 *SenseModule* Position, Alignment, and Calibration

The position and alignment of the *SenseModule* within the finger hole can each have a significant impact on the acceleration readings, and thus on the calculations that are derived from those readings. Further, the ADXL345 introduces some additional alignment considerations of its own.

A fixed or known depth of the *SenseModule* within the finger hole is crucial to deriving accurate angular velocity calculations from the magnitude angular acceleration readings. The centripetal acceleration that the ADXL345 experiences is related to its radius of rotation, which is not the radius of the ball, but rather, the offset of the ADXL345 from the center of the ball, and/or the actual axis of rotation. In the final round of data collection conducted for this paper, the ADXL345 was located 1.475" below the surface of the ball, putting it at a distance of 2.80" from the center of the ball, thus r = 2.80" for the angular velocity calculations.

The pitch of the finger hole also comes into play, as the holes drilled in a bowling ball are not necessarily directed toward the center of the ball, but are pitched in such a way as to aid the bowler in applying lift to the ball (finger holes pitched toward the palm, and thumbhole pitched away from the palm). Also, bowling balls are routinely drilled to introduce a predictable dynamic imbalance in the ball, intended to manipulate the ball's moment of inertia, and either resist or enhance the onset of hook in the ball. Such drilling patterns shift the radius of gyration of the ball away from its center of mass.

The *SenseModule* does not yet have a case to secure it within the finger hole. For the data collected for this project, the *SM* was secured to the bottom of the finger insert with tape. As such, the finger insert transferred some of the forces it experienced during approach and release to the *SM*. It is also likely that the higher frequency noise imposed on the REACTION segment waveform is due to vibration and excess movement transferred to the *SM* from the movement of the module and the finger insert.

In order for the *SenseModule* to provide consistent and repeatable results, it must be positioned and aligned within the finger hole in a consistent and repeatable manner. Thus, a case must be developed that can be secured in the bottom of the finger hole which will then fix the depth and rotational alignment of the *SM* for a particular bowling ball. With such a case, the *SM* could then be calibrated for that ball, with a known alignment in the ball – as shown in Figure 25.

5.6.4 External Forces and Friction

The major assumption we've made is that the frictional force between the ball and the lane is the only force of any significance that acts on the ball following release. Losses due to noise, vibration, aerodynamic drag, and torque applied to the rotational plane of the ball are considered negligible. A further assumption is that the force due to kinetic friction acts on the ball in such a way as to efficiently transfer its linear kinetic energy to angular kinetic energy.

The ball is released with an initial linear velocity that exceeds the ball's angular velocity. That is, the ball travels further during one revolution of the ball than one circumference of the ball. This difference causes the ball to skid (slide). Also, generally speaking, the axis of rotation of the ball is neither parallel to the surface, nor is it normal to the direction of the lane. The force due to friction causes the ball to resolve those differences, slowing the ball down, while increasing its angular velocity, and causing the ball to hook. That resolution continues to occur until such time as the ball is no longer skidding (has rolled out), or the ball has completely traversed the lane and entered the pit.

- **The ball is always slowing down after it is released:** The force due to friction causes the ball to translate linear kinetic energy to angular kinetic energy, until such time as the resolution (rollout) point is reached. If the ball reaches rollout, both the angular and linear velocities decrease. Therefore, the linear velocity is always continuously and monotonically decreasing.
- **The angular velocity is always increasing:** The angular velocity of the ball will continuously and monotonically increase until the rollout point is reached, at which time the angular velocity also starts to decrease in a continuous and monotonic fashion, along with the linear velocity. It is possible to detect roll out by observing the angular velocity of the ball near the pins. The ball has rolled out if the angular velocity has begun to decrease, in which case, the linear velocity of the ball can be directly deduced, since it must travel 27" (one circumference of the ball) for every revolution of the ball.
- **Perfect transfer from linear kinetic energy to angular kinetic energy occurs:** Actually this is not the case, but the amount of transfer can be detected in the change in angular velocity between each revolution of the ball. Since the kinetic frictional force is the only force acting on the ball, it will be directly related to the amount of change in the angular velocity. The actual unaccounted for losses due to friction result from heating of the ball and/or the lane, noise generation, vibration, inelastic deformation of the ball and lane, and torque applied to rotating the axis of the ball normal to the direction of travel.

If the ball rolls out, it is possible to directly observe the effects of rolling friction, since the linear velocity is directly related to the angular velocity, and the drop in kinetic energy of the ball can then be measured. While the ball is skidding, the actual losses due to friction are very probably some constant fraction (percentage) of the change in angular momentum, i.e., 90% of the change in linear momentum is transferred to angular momentum, 10% is given up as heat, noise, etc. The exact percentage is not yet known, and it probably varies for each type of ball, but with additional research, it should be possible to place an upper and lower bound on the frictional losses.

5.7 Waveform Analysis Future Work

Although the *SenseModule* is now fully functional, in that it currently operates in a fully autonomous mode, there is still much work to do. The future work for the *SM* has already been proposed earlier in the paper, but there is also much additional work that must be done on the data analysis, algorithm development, and validation and verification fronts.

5.7.1 Bowling Metric Accuracy

Although extensive work has been done to develop the bowling metric extraction algorithms, little has been done to verify the accuracy of their output. The author has used standard video analysis to verify that the average linear velocity, the revolution count, and the angular velocity calculations are approximately correct, within the limitations of that method of analysis. However, video analysis, at 30 fps, does not provide the requisite temporal resolution necessary to evaluate the instantaneous linear and angular velocity calculations, nor the distance calculations.

The author has used the same measurement techniques as with the first paper. With markers applied at known distances on the lane, and additional markers applied to the ball each 60° of rotation, multiple shots have been simultaneously captured with the *SenseModule* and recorded on a digital camcorder.

For an average ball speed of 15 mph (22 fps), the ball travels about 9” per frame, more at the start, less at the pins. The error margin is twice that value (the first and last video frames), so that the best error margin is ±2.5% (or about 0.4 mph), and the error margin increases as the velocity increases. A similar scenario exists when counting revolutions. For a rotation rate of 6 Hz (360 RPMs), the resolution is 72 degrees of rotation at 30 fps (or about 0.2 revolutions). Again, the error margin increases with increasing RPMs. In either case, it is pointless to try to assess the instantaneous linear and angular velocities without using some form of high-speed video camera, with at least 10 times the frame rate.

Recall that the *RevMetrix* system is intended to provide the bowler with metrics useful for evaluating their execution, provide information useful for evaluating the reaction of the bowling ball to the lane condition, as well as provide insight into changing lane conditions.

As such, it is more important that the *RevMetrix* system provide consistent, repeatable results, rather than absolutely accurate results. Ideally, if the bowler executes their delivery the same way on two different shots, and the ball responds identically to that stimulus, then the system should report similar results. On the other hand, if the bowler executes consecutive shots with a 5% difference in release RPMs between them, the system should report a 5% difference in release RPMs.

At some point, it will be necessary to verify the metrics that the *RevMetrix* system produces. The current options have changed little in the time since the author first addressed this problem a dozen years ago:

- Contact the USBC to use their automated bowling robot (E.A.R.L.) to throw repeated shots (with a bowling ball equipped with a *SenseModule*) at known linear and angular velocities, along with predetermined axis turn and tilt angles [11].
- Contact Brunswick to obtain time on their "Throbot" machine, similar to USBC's E.A.R.L. [12].
- Visit a CATS-instrumented facility and correlate the *RevMetrix* analysis results against the CATS findings [13].

5.7.2 Approach Characteristics

No attempt has been made to analyze the acceleration data that the *SenseModule* collects during the bowler's approach. Since the ball is not rotating during approach, that data should reveal the position of the ball in the bowler's stance, as well as the motion of the bowling ball as the bowler delivers the ball to the lane. That approach data could reveal the speed of the bowler's delivery, the relative height of their back swing, and their possible release point above the lane. Extensive data collection and analysis research, along with high speed video comparisons would be necessary in order to create a quantitative analysis of the bowler's approach characteristics. On the other hand, that data could be used to develop a qualitative approach "signature" that could reveal the consistencies/inconsistencies in the bowler's approach, and the effects of changes that the bowler is attempting to make in their approach.

5.7.3 Release Characteristics

As part of the approach data, the *SenseModule* also captures the bowler's release of the bowling ball. Recall that it is during release that the bowler applies lift and turn to the ball, as well as applies the impetus to loft the ball, and give the ball its initial linear and angular velocities. It is the release motion that determines how the ball will react once it hits the lane. It should also be possible to quantify the release motion and create a release "signature" for the bowler to inspect. This would also be helpful in identifying variability in the bowler's release, as well as to provide feedback into any adjustments they might be attempting to make to their release. In addition, if it is possible to obtain the orientation of the ball at the time of release, it would then be possible to identify the axis turn at release, which would reveal the initial direction of rotation with respect to the lane (direction of travel).

5.7.4 Axis Tilt Angle

The phases and amplitudes of the separate 3-axis tilt responses should reveal the tilt of the axis of rotation as the ball rolls down the lane. The tilt response traces a sinusoidal waveform with maximum amplitude of ± 1 *g*. If the axis of rotation is tilted away from parallel with the lane surface, that amplitude will drop in direct proportion to the *sine* of the angle of tilt. The 3-axis tilt response can be used to reveal how the tilt of the ball's rotational axis changes over time during the REACTION segment.

Section VI: Summary

The *SenseModule* developed for this paper is a prototype, believed to be the first of its kind (low cost, low power, low mass, unobtrusive, autonomous operation). As such, this has been a project of broad scope and reach. The *SenseModule* was first developed as a data collection platform in order to discover the content and morphology of the waveforms it was intended to capture. It took several iterations of hardware and embedded software before the *SM* achieved autonomous operation.

At that point, the process of collecting and analyzing data began, with most of the analysis performed using MATLAB. Once the morphology of the raw acceleration data waveforms became apparent, wavelet decomposition/reconstruction was identified as the best method for determining the boundaries of the various morphological segments in the data. It was then possible to implement an algorithm for automatically dividing the waveform into its constituent segments (APPROACH, LOFT, REACTION, and IMPACT).

Each segment can be characterized by its frequency content and its shape (how the signal evolves over time). Certain segments lend themselves to analysis by finite impulse response (FIR) filtering, others are better viewed through the lens that Wavelet Theory provides, and yet others require a combination of both techniques.

The APPROACH segment is best analyzed with wavelets, as it has a low-frequency, non-cyclical shape that evolves from DC to perhaps a single 0.5 Hz cycle generated by the bowler's arm swing, and culminates in a sudden increase in angular velocity as the bowler applies lift and turn to the ball.

The LOFT segment immediately follows release and is also best analyzed with wavelets, as it is dominated by flat DC content, along with one or more brief significant spikes that result from the ball impacting the lane (and possibly bouncing) after being lofted during release. The DC portion of the LOFT segment supplies an isolated view of the centripetal acceleration of the ball as the ball is rotating in free fall during loft.

The REACTION segment commences once the ball remains in continuous contact with the lane, and is dominated by a noisy sinusoidal frequency chirp superimposed upon a significant DC component generated by the centripetal acceleration of the ball. The frequency chirp is the result of the tilt aspect of the ADXL345, and gives a direct indication of the orientation of the *SenseModule* with respect to the lane surface as the ball rolls down the lane. The REACTION segment is best analyzed using FIR techniques to filter out the low frequency acceleration component and high frequency noise from the tilt response. It is that filtering of the REACTION segment that allows us to extract the changing angular velocity of the ball. The angular acceleration component can also be used for that same purpose.

The IMPACT segment begins with the ball's initial encounter with the pins, and continues through the ball falling off the end of the lane into the pit. This segment is characterized by a continued tilt response imposed on a low frequency component, but with numerous significant spikes due to pin impacts, and greater noise content, also due to the ball driving through the pins. Both the linear and angular velocities of the ball slow down significantly and the segment terminates with a period of free fall (flat

response) as the ball falls into the pit. Analysis of the IMPACT segment requires both FIR and wavelet techniques, due to its varied signal content.

Relying on an assumption of energy conservation, we developed automated algorithms that isolate the angular velocity tilt response from the REACTION segment, and then used the angular acceleration response of the LOFT segment to create a combined LOFT-REACTION instantaneous angular velocity waveform. From there, we extracted the individual revolutions of the ball, and inferred the instantaneous linear velocity from the changes in angular velocity using the efficient transfer of linear kinetic energy to angular kinetic energy. Finding the instantaneous linear velocity required the development of an iterative converging algorithm that compares the expected distance the ball travelled with the distance that results from the linear velocity "guess" we had just calculated.

Having found the instantaneous linear velocity of the ball, it then became possible to deduce the location of the ball with respect to the foul line, and thus the distance the ball was lofted. We could then also locate each revolution with respect to the foul line, which then allowed us to establish the *break point* of the ball. Finally, we were also able to derive the kinetic frictional force from the changes in either the angular or linear velocities.

The author has taken great liberty in assuming that no energy is lost during the transfer of energy from the ball's translation to the ball's rotation. The analysis algorithms presented here are just a first attempt at showing what is ultimately possible from having collected the 3-axis accelerometer data from a sensor positioned underneath a finger hole in a bowling ball.

The author has yet to resolve the discrepancy that arose between the angular velocity extracted from the tilt response waveform and the corresponding angular velocity extracted from the isolated angular acceleration component of the combined LOFT-IMPACT segment. The ADXL345's response during the LOFT segment is the result of the isolated centripetal acceleration and should lead directly to the angular velocity of the ball. The peak-valley and valley-peak tilt response should also lead directly to the angular velocity of the ball for each half-rotation. However, those results disagree by about 5%, and that error seems to be out of line with the accuracy we should be able to expect from the system.

It must be stated that all of the waveforms used for development of the MATLAB segmentation, extraction, and analysis algorithms have originated with the author's use of the *SenseModule*. As such, data collection with the *SenseModule* across a wider range of bowlers and bowling styles is still necessary, which will undoubtedly result in further development and refinement of the algorithms presented in this paper. Eventually, formal testing must be conducted in order to verify and validate the accuracy of the analysis algorithms.

Admittedly, there is much additional *SenseModule* refinement, raw data collection, waveform analysis, and algorithm development that has yet to be accomplished, and objective verification and validation must also be performed across the entire *REVMETRIX* system. However, the project, to this point, has reached its goal of developing an autonomous, unobtrusive *in situ* bowling ball sensor module, coupled with an automated analysis system that provides quantitative feedback to the bowler about their execution, as well as the reaction of the ball once they have released it to the lane.

REFERENCES

Books and Literature

[1] Hake, “A Performance Analysis System for the Sport of Bowling”, Computer Science Master’s Paper, Penn State Harrisburg, May 2002

[2] Vint, "Bowling's Invisible Obstacle Course", *Bowling Magazine*, December/January 1993

[3] Fuss, Kong, and Tan, “The First Instrumented Bowling Ball”, Nanyang Tech University, March 2007

[4] Fuss, Khang, “Performance Analysis with an Instrumented Bowling Ball”, The Impact of Technology on Sport II, Taylor and Francis, 2008

[5] Frohlich, “What Makes Bowling Balls Hook”, American Journal of Physics, Vol. 72, No. 9

[6] Talamo, “The Physics of Bowling Balls”, www.docshut.com, 2008

[7] King, Perkins, et al, “Bowling ball dynamics revealed by miniature wireless MEMs inertial measurement unit”, ISEA 2011, No. 13, pp 95-104

[8] The Physics of Bowling, www.real-world-physics-problems.com, 2014

[9] Nave, HyperPhysics, Dept. of Physics and Astronomy, Georgia State University, 2014

[10] USBC Equipment Specifications Manual, www.bowl.com, March 2014

[11] USBC's E.A.R.L. Bowling Robot, USBC, 2014

[12] Brunswick's "Throbot" Bowling Robot, Brunswick Corporation, 2014

[13] C.A.T.S. - Computer Aided Tracking System, Kegel Training Center, 2005

[14] Moore, *MATLAB for Engineers*, 3rd Edition, Pearson, 2012

[15] Schilling and Harris, *Fundamentals of Digital Signal Processing*, Thomson, 2005

[16] Strang and Nguyen, *Wavelets and Filter Banks* , Wellesley-Cambridge Press, 1997

[17] Burrus, Gopinath, and Guo, *Introduction to Wavelets and Wavelet Transforms*, Prentice Hall, 1998

Software Packages

[18] *PK51 Professional Developer’s Kit* , Keil Software, February 2010

[19] *Eagle PCB Software V5.0*, Cadsoft USA, 2008

[20] *MATLAB Student Version R2012a*, The Mathworks, 2012

[21] *MATLAB Signal Processing Toolbox*, The Mathworks, 2012

[22] *MATLAB Wavelet Toolbox*, The Mathworks, 2012

Hardware Components

[23] C8051F92x-F93x Ultra Low-Power MCUs, Silicon Laboratories

[24] 24FC1025 128K x 8 Serial EEPROM, Microchip Technologies

[25] ADXL345 3-Axis ±16 g Digital Accelerometer, Analog Devices

[26] TSL13T Light-to-Voltage Converter, AMS Sensor Solutions

[27] Optek OP521 Phototransistor Datasheet, Optek Corporation

[28] Manganese Dioxide Lithium Batteries (CR series), Panasonic Corporation

[29] Si1141 Proximity and Ambient Light Sensor, Silicon Laboratories

Prototype Services

[30] Quick-Turn Prototype and Small Quantity PCBs, Advanced Circuits

[31] Quick-Turn Prototype and Small Quantity SMT Assembly, Advanced Assembly

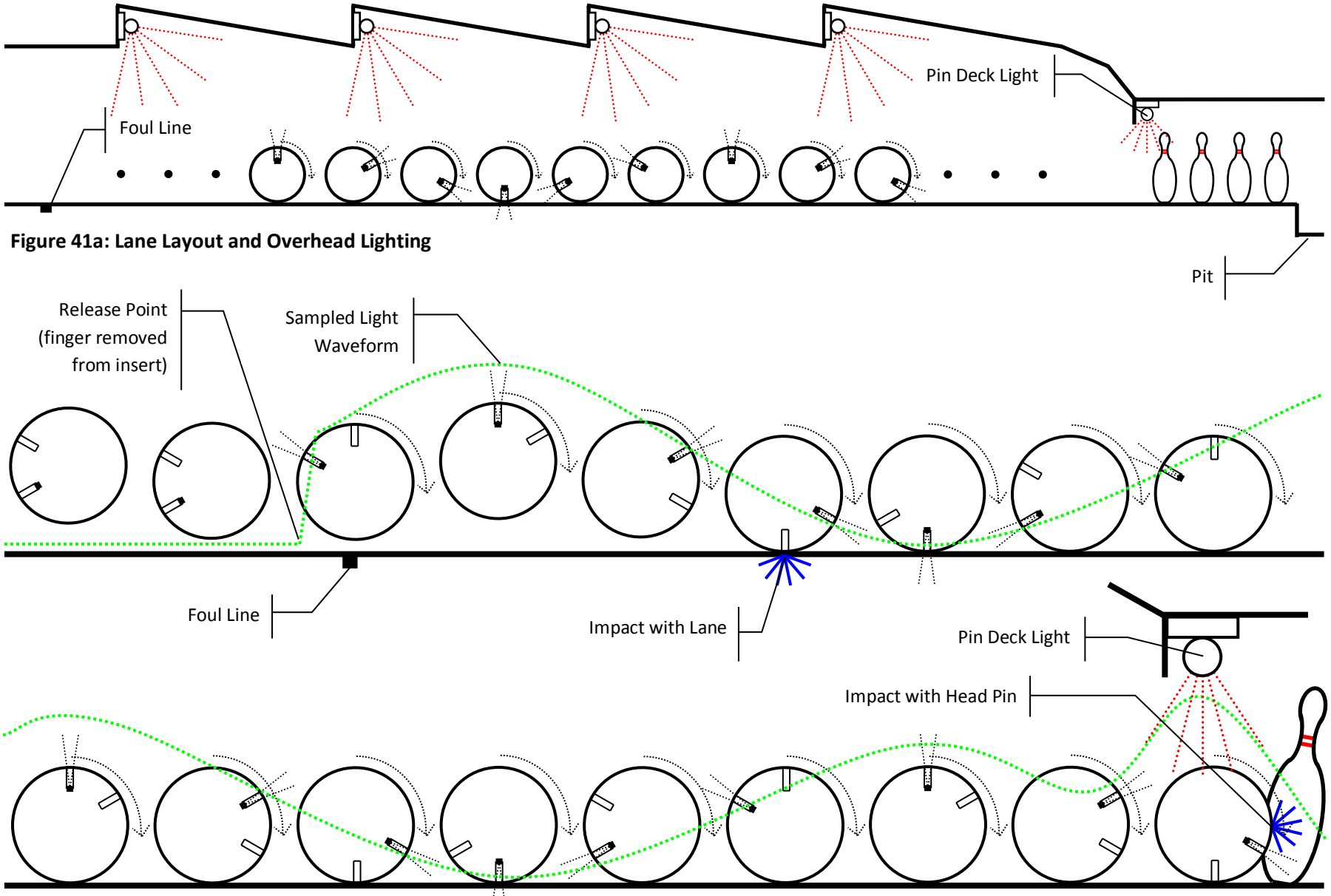

Figure 41a: Lane Layout and Overhead Lighting

Figure 41b: Ambient Light Waveform

Figure 41: Lane Layout and Ambient Light Waveform

APPENDIX B: *SMARTDOT* AND *SENSEMODULE* LIGHT AND IMPACT COMPARISON

Figure 42 is a typical *SMARTDOT* module graph. The original *SMARTDOT* module from [1] collected ambient light data (TSL251 LTV converter) and impacts (piezoelectric film). The *SenseModule* also collects ambient light data (TSL13T LTV converter), along with 3-axis acceleration data, and Figure 43 is a typical *SenseModule* graph, with impact locations added in red. Although the ambient light data was collected 16 years apart, using two separate module designs with similar LTV converters, but completely different technologies for impact detection, the light and impact data are remarkably similar.

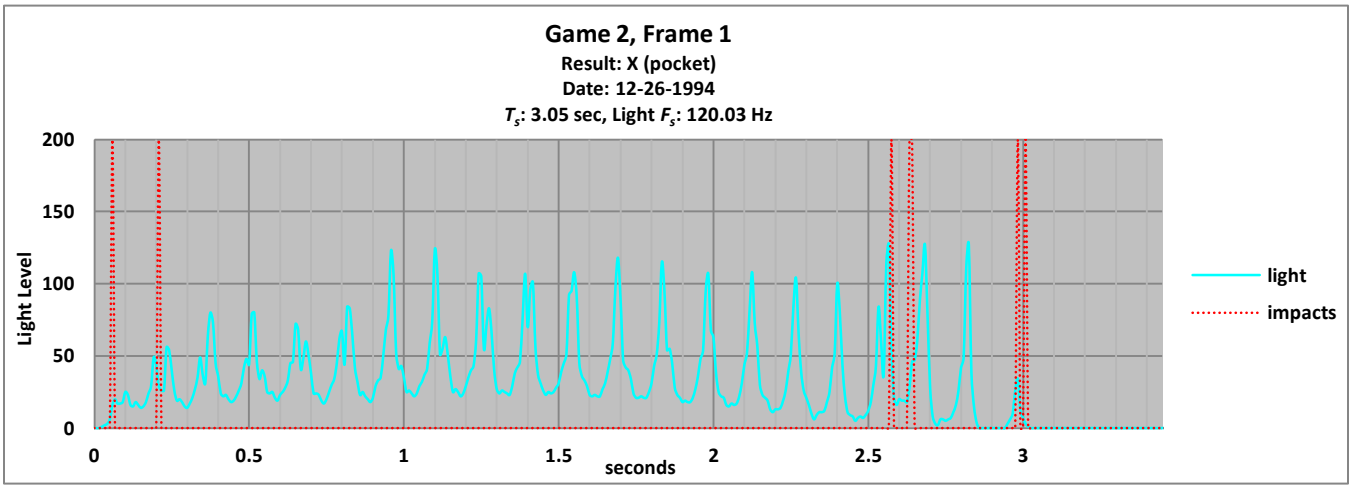

Figure 42: Typical *Smartdot* Ambient Light Waveform with Impacts

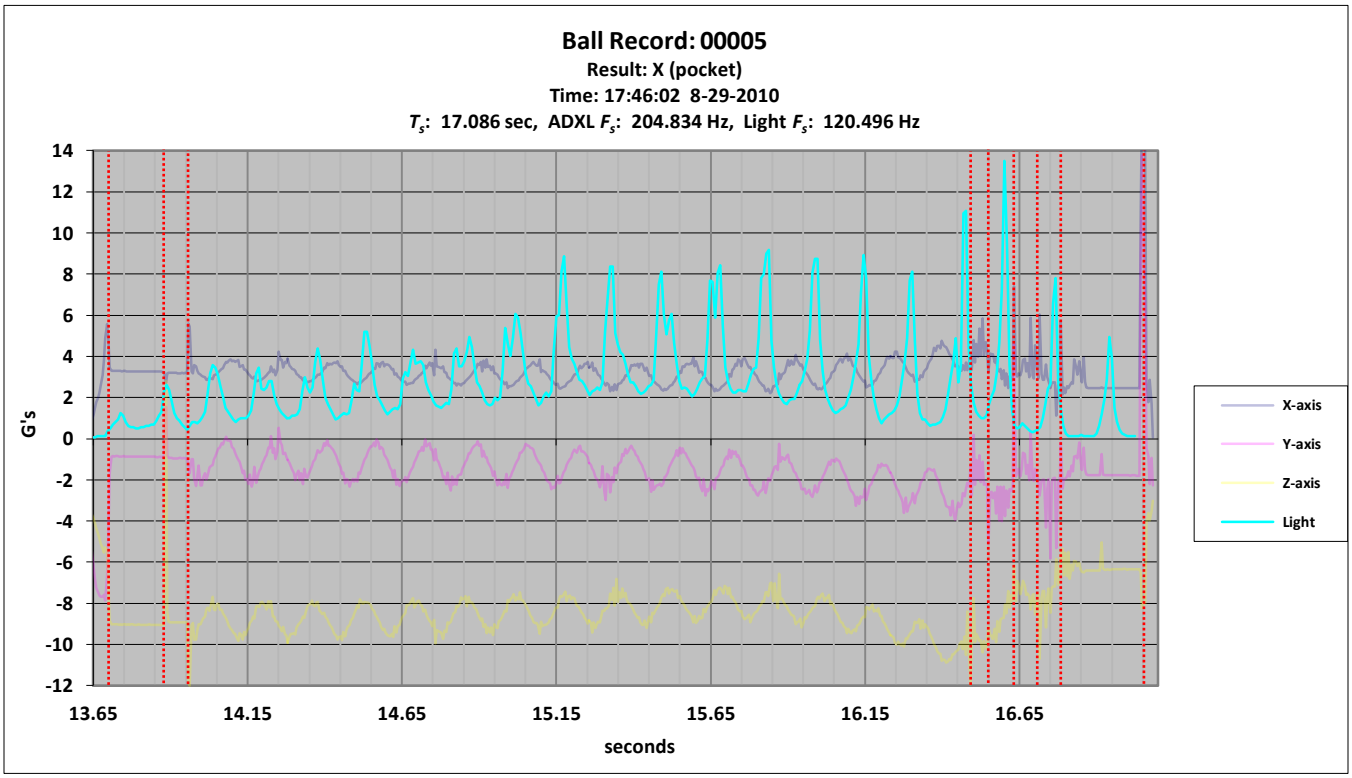

Figure 43: Typical *SenseModule* Ambient Light Waveform with Impacts

APPENDIX C: *COMMODULE* COMMUNICATION PROTOCOLS

ComModule Detection Protocol

A means for exchanging information with the *SenseModule* is required in order to program the module, configure the module, and upload sensor data from the module. The TSL13 light-to-voltage converter also serves as the receiver for the *SM* communication circuit. The sensor module receives serial commands and data via the TSL13, and transmits serial responses via the transmit LED. The bowler will initiate a communication sequence with the sensor module by placing the *ComModule* over the finger hole, generally while the module is in standby mode. Thus, the act of placing the communication module over the finger hole causes the module to go through the standard wake-up sequence. The *ComModule* detection protocol is given in the following sequence of steps, which describes the timing diagram shown in Figure 44.

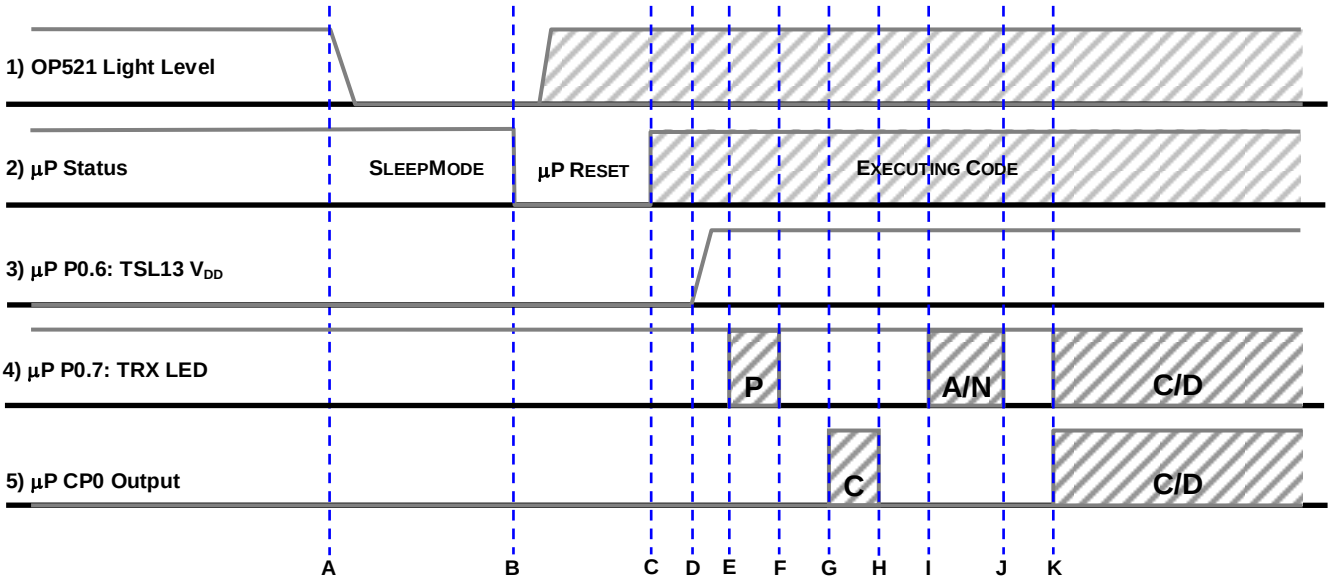

Figure 44: *ComModule* Detection Timing Diagram

A) The bowler places the *ComModule* over the finger hole containing the sensor module, blocking ambient light from reaching the start-up circuit.

B) The minimum start-up dark-time is reached (nominally 500 ms), and the µP enters reset.

C) The µP exits reset and begins executing its self-configuration code.

D) As part of its embedded program, the µP first "assumes" that it is receiving a transmission from the *CM*. It applies power to the TSL13 through P0.6, and configures comparator CP1 to detect positive edges (which are the leading edges of logic '0' bits) generated by the TSL13. The CP1+ input is P1.0 and the CP1- input is P1.1.

E) After the μP is configured, it transmits the "probe" character sequence, indicated by 'P' in line 4 of Figure 44. The detailed byte transmission timing diagram is given in Figure 46.

F) The μP clears any transient interrupts generated at CP1 due to applying power to the TSL13, and then enables the CP1 positive edge interrupt in order to detect the leading edges of logic '0' bits transmitted by the *CM*. The detailed byte reception timing diagram is given in Figure 45.

G) At the conclusion of the probe transmission, the μP switches to receive mode, and waits for a command character sequence ('C' in line 5) from the *CM*.

H) If the command sequence starts within the allotted time (nominally 100 ms), the μP receives the command sequence. If the μP does not detect a command sequence within the allotted time, it issues the probe sequence up to two more times. If the presence of the *CM* is not detected, the μP switches to sampling mode.

I) At the end of the command (detected via a terminating character or a time out value), the μP checks the validity of the command through a checksum test. If the checksum is valid, the μP waits for the communications module to enter receive mode (typically one character time).

J) The μP then transmits an "ACK" (acknowledge) for a valid checksum, or a "NAK" (negative acknowledge (invalid checksum), switches to receive mode, and waits for the next transmission from the *CM*.

K) If the μP received a valid command, communication then proceeds between the two modules according to the specific command protocol. If an invalid command was received, the μP waits to receive the retransmission of the command.

Serial Reception Protocol (Infrared iRTZ UART)

The TSL13 light-to-voltage converter doubles as the infrared serial receiver. Its output pin is tied to P1.0, configured as the positive (CP1+) input of comparator CP1, while the negative (CP1-) input is tied to a voltage divider set to a level appropriate for detecting light pulses in a dark ambient environment. The serial reception scheme utilizes a modified UART protocol, with one START bit, 8 DATA bits, and 1 STOP bit. An inverted Return-To-Zero (iRTZ) format is used, with reception occurring under ambient dark conditions, where each "space" (0 bit) is transmitted as a light pulse consisting of a rising edge, a minimum duration (1 μs), and a falling edge, while the absence of light is considered to be the "mark" (1 bit) level.

Two μP resources are used to implement the byte reception function: comparator CP1 is configured as the START bit detector and an auto-reload timer is configured as the bit slice timer (BST). The CP1 interrupt detects the leading (rising) edge of the START bit, and the BST interrupt signals the individual bit slice times. Before the anticipated reception of each byte, the BST is halted, the BST interrupt is disabled, the BST value is initialized to 1.5 bit times, while the BST auto-reload value is initialized to 1 bit time. Thus, once the BST is enabled at the rising edge of the START bit, all BST interrupts occur in the middle of their respective bit slice times. The BST performs the reception of bits.

By checking for bits in the middle of the bit time, the serial receiver can overcome differences between the baud rate generators of the transmitter and the receiver due to temperature drift, calibration error, and time-base resolution errors. This scheme will accommodate combined errors of up to ±5% per bit time – accumulating a ±47.5% drift across the 9.5 bit times it takes to receive the 8 DATA bits and the single STOP bit. The error margin can be increased to ±6% per bit time if 2 STOP bit times are guaranteed to be transmitted between bytes.

At each BST overflow, the BST ISR reads the CP1 rising edge flag. If the flag is set (1), then a rising edge was detected during the preceding bit time, and a '0' bit is placed at the corresponding position in the serial byte being assembled. If no rising edge was detected, a 1 bit is placed in the serial byte bit position. After 8 data bits have been received, the next BST overflow captures the STOP bit as a flag. After the STOP bit time, the BST ISR disables the BST interrupt, and a flag is set to indicate that a byte has been received. Any routine that processes the received byte can then check the validity of the STOP bit.

Figure 45 presents the timing diagram for the serial reception scheme.

UART receiver settings:

START bit:	1
DATA bits:	8
PARITY bit:	NONE
STOP bits:	2 (allows for additional inter-character process time)
BAUD rate:	28,800 kbaud, individual bit time = 34.72 μs.
Data transfer rate:	~2500 bytes/sec (assuming 11 bit frame)

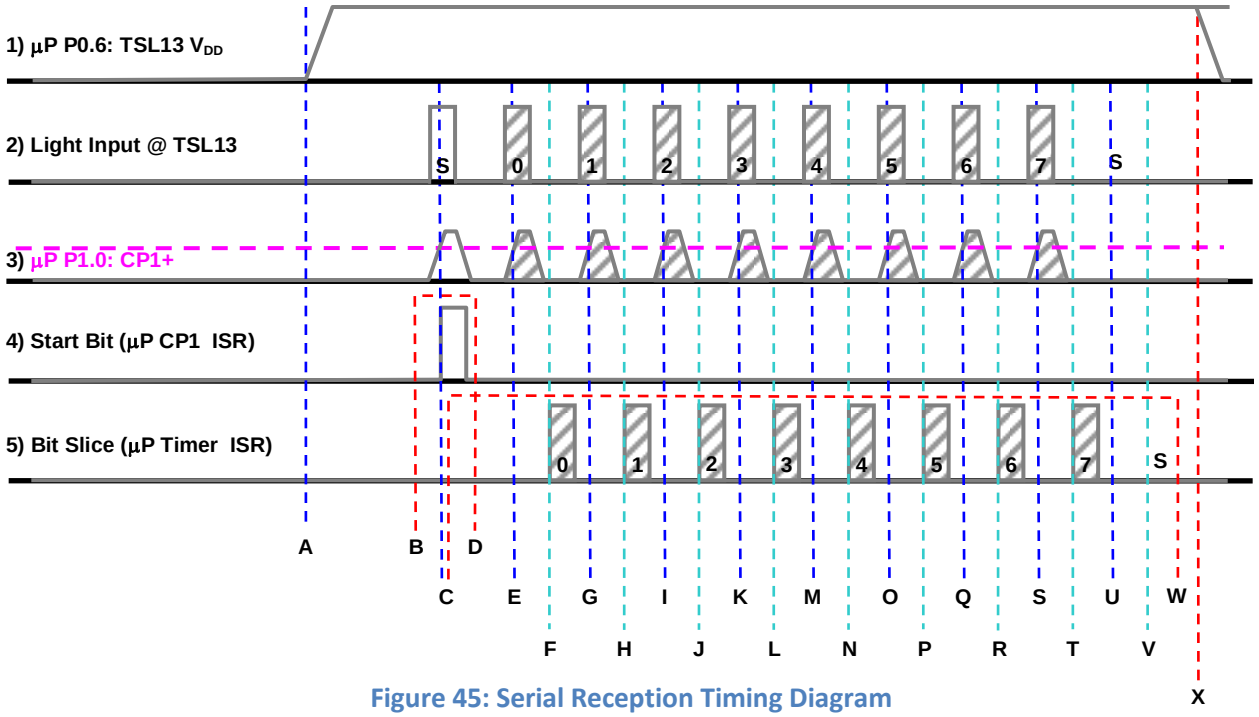

Figure 45: Serial Reception Timing Diagram

A) Before serial reception can begin, the µP must supply power to the TSL13 via P0.6.

B) After the TSL13 output settles (~100 µs), the µP enables the CP1 rising edge interrupt. This should occur in an ambient dark condition, as measured with the TSL13. The CP1 interrupt remains enabled from point A to point D.

C) The first rising edge at CP1+ causes a CP1 interrupt – the rising edge of the START bit. The CP1 interrupt service routine (ISR) starts the BST and enables the BST ISR. The BST interrupt and timer remain enabled from point C to point W.

D) The CP1 ISR disables the CP1 interrupt.

E) Even with the CP1 interrupt disabled, rising edges are still detected and the CP1 rising edge flag continues to be set accordingly (points E, G, I, K, M, O, Q, S).

F) At each BST overflow, the BST ISR checks the CP1 rising edge flag and stores the appropriate value (a '0' bit if a rising edge occurred during the preceding bit time, and '1' bit otherwise) in the byte being assembled. This occurs for each of the 8 DATA bits at points F, H, J, L, N, P, R, and T. The ISR also clears the CP1 rising edge flag for the next bit time.

U) After the 8 DATA bits have been collected, the next BST overflow is for STOP bit detection. There should be no rising edge transition during this bit time.

V) The BST ISR sets the STOP bit flag – '1' for a valid STOP bit (no CP1 rising edge), '0' otherwise.

W) The BST ISR disables the BST interrupt, and sets a flag indicating that a byte has been received.

X) If another byte is anticipated, the process repeats from step B, otherwise, the µP removes power from the TSL13.

Serial Transmission Protocol (Infrared iRTZ UART)

Since the *SenseModule* can transmit the entire contents of its sample memory (128 kbytes) during one request, it is beneficial for the *SM* to transmit at a much higher rate than it can receive. The *ComModule* must then have an infrared reception circuit that can accommodate the faster *SM* transmission rate.

The serial transmission scheme utilizes the modified UART protocol given for the serial reception mode, with one START bit, 8 DATA bits, and 2 STOP bits. An inverted Return-To-Zero (iRTZ) format is used, with transmission occurring under ambient dark conditions, where each "space" (0 bit) is transmitted as a light pulse consisting of a rising edge, a minimum duration (50% duty cycle), and a falling edge, while the absence of light is considered to be the "mark", or logic '1', bit level. The light pulses are 50% of the bit time, which reduces the overall current required during serial transmission.

A single μP timer is used to implement the byte transmission function - an auto-reload timer configured as the bit slice timer (BST). The BST triggers the start of each bit time. The BST ISR then outputs the next bit. If it is a '0', the ISR turns the transmit LED on, and then turns the LED off after a fixed delay time. If the bit is a '1', there is nothing to do, since the LED is already off.

Since the μP does not have to contend with multiple tasks when it is transmitting data, the transmission of serial data is accomplished with a simple "bit-banging" scheme, with the BST trigerring the bit times, and the byte transmission routine simply waiting for the BST overflows. By utilizing an optimized "bit-banging" technique, it is possible to implement a *SenseModule* transmission rate of up to 115.2 kbaud.

Figure 46 presents the timing diagram for the serial transmission mode.

UART transmitter settings:

START bit:	1
DATA bits:	8
PARITY bit:	NONE
STOP bits:	2 (allows for additional inter-character process time)
BAUD rate:	28,800 to 115.3 kbaud, individual bit time = 8.68 μs to 34.72 μs.
Data transfer rate:	~2500 to ~10,000 bytes/sec (assuming 11 bit frame)

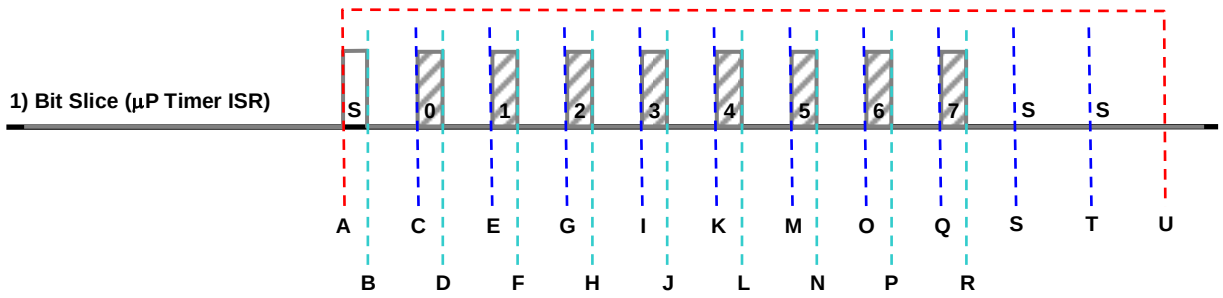

Figure 46: Serial Transmission Timing Diagram

A) *SenseModule* initiates transmission by passing the byte to be transmitted to the low-level transmit routine, enabling the Bit Slice Timer (BST) overflow interrupt, preloading the BST to immediately overflow, and then starting the BST. The BST ISR issues the START bit (a logic 0) by turning on the the current source (IREF0) at P0.7.

B) After a fixed delay, the BST ISR terminates the START bit (returning to "dark"). The active duty cycle of the transmit LED is 50% when issuing a light pulse, and is 0% when issuing a dark pulse.

C) At the next BST overflow, the BST ISR issues the first DATA bit (bit 0) – a light pulse for a logic 0, or remains dark for a logic 1. DATA bits are issued at points C, E, G, I, K, M, O, and Q)

D) The BST ISR terminates a '0' DATA bit by shutting off current to the transmit LED - points C, F, H, J, L, N, P, and R).

R) The BST ISR that issues the last DATA bit also sets the BST for two consecutive bit times, which will cause two STOP bits to be issued following termination of the last DATA bit, and the expiration of the last DATA bit time – points R, S, and T.

U) At the next BST overflow, the BST ISR halts the BST timer, disables the BST, resets the transmit variables for the next byte, and sets a flag to indicate that the data byte has been transmitted. Note that the current drive (IREF0) for the transmit LED has already been turned off by this time.

APPENDIX D: TYPICAL MATLAB OUTPUT METRICS

```
Enter data set name ('mmddyy-nnnnnn'): '082910-00009'

  Light RELEASE detected at Light index 231, TS = 11.872192 secs
  All time stamps adjusted relative to Light RELEASE TS

  RELEASE impact at ADXL index 687, TS = 0.023420 secs
  All time stamps adjusted to ADXL RELEASE point

  PIN impact at ADXL index 1248, TS = 2.739363 secs
  Average ball speed = 14.934 mph (21.903 fps)

  LOFT impact 1 at ADXL index 729, TS = 0.205086 secs
  Approximate distance = 4.49 ft

  LOFT impact 2 at ADXL index 744, TS = 0.278331 secs
  Approximate distance = 6.10 ft

  Release angular velocity(FIR): 357.9 rpms
  Impact angular velocity(FIR):  423.7 rpms
  Total Revolutions(FIR):         15.0

  Average linear velocity:  14.93 mph
  Release linear velocity:  15.00 mph
  Impact linear velocity:   14.57 mph

  Loft Distance:      54 in (4.51 ft)
  Reaction Distance:  76 in (6.34 ft)
```

APPENDIX E: TYPICAL RAW DATA WAVEFORMS

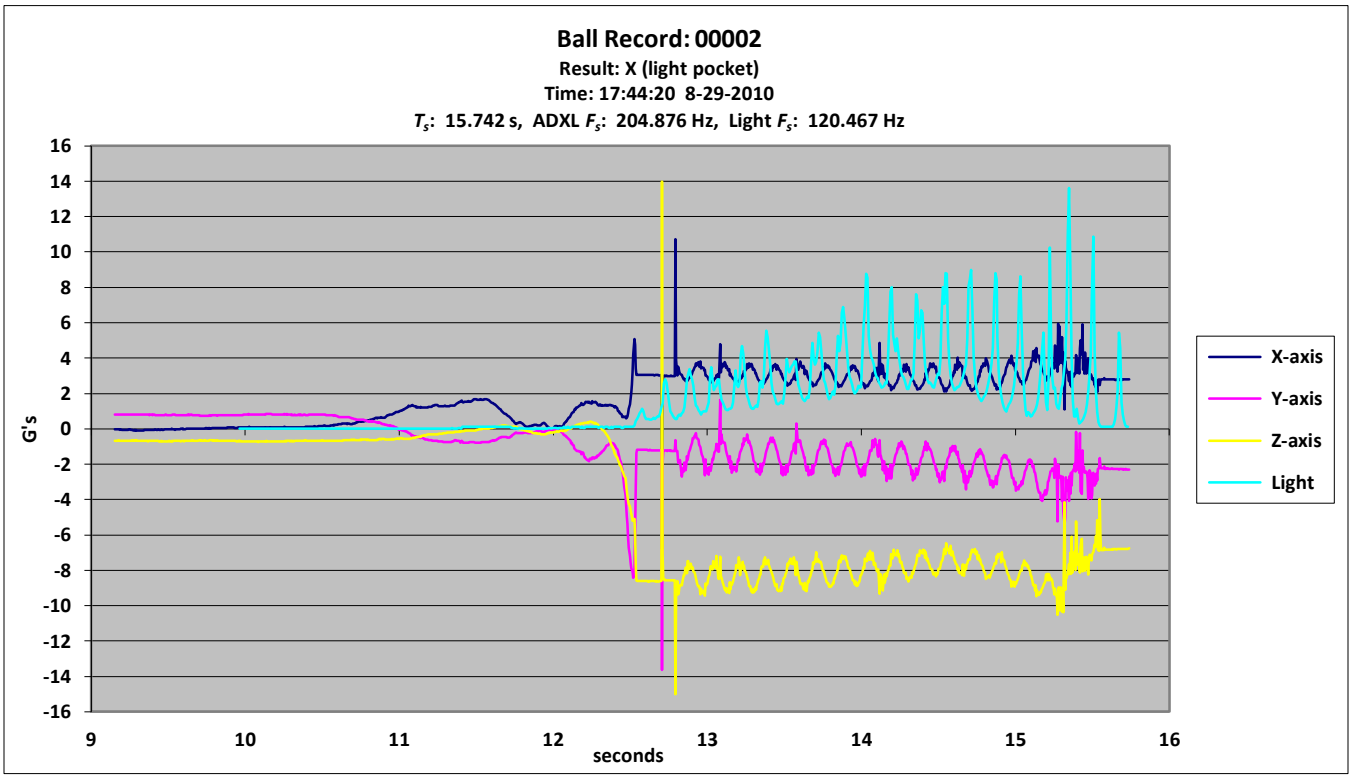

Figure 47: Ball Record 00002 (typical waveform)

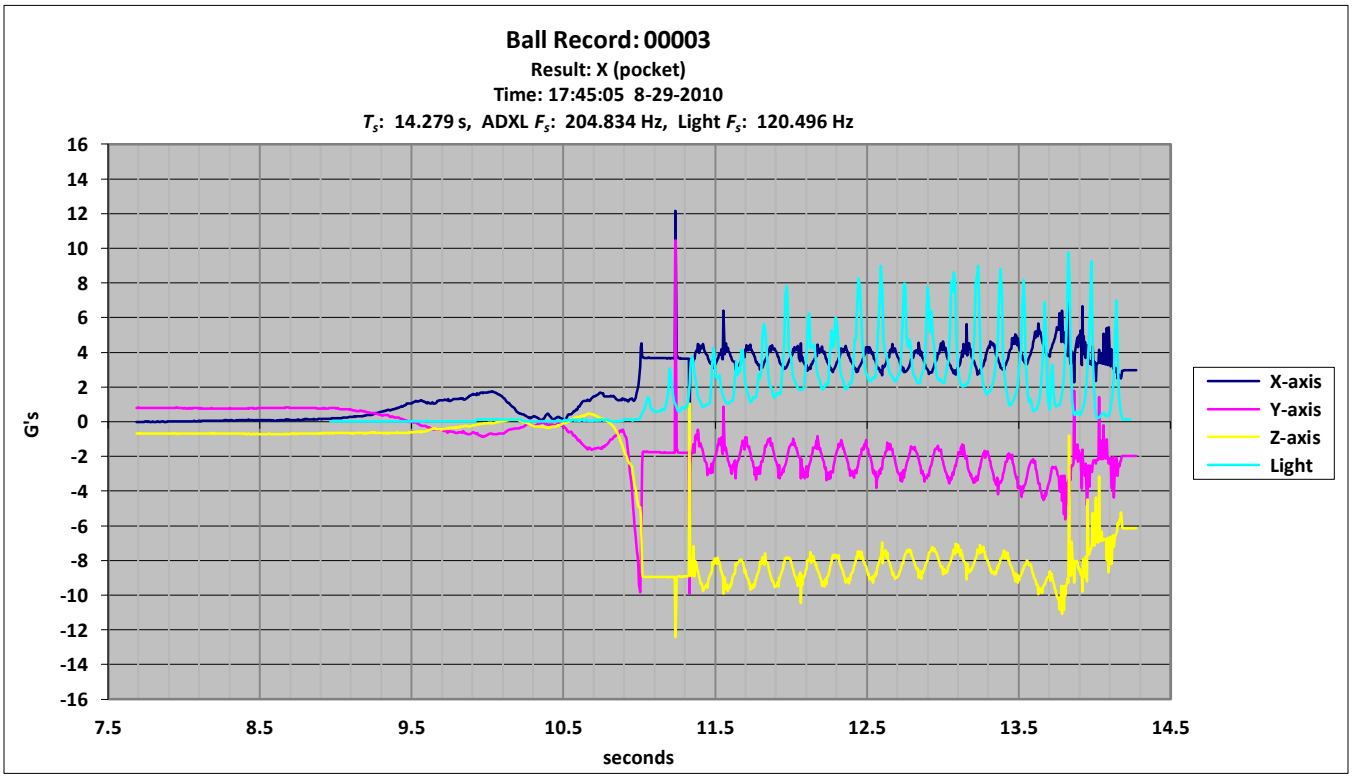

Figure 48: Ball Record 00003 (typical waveform)

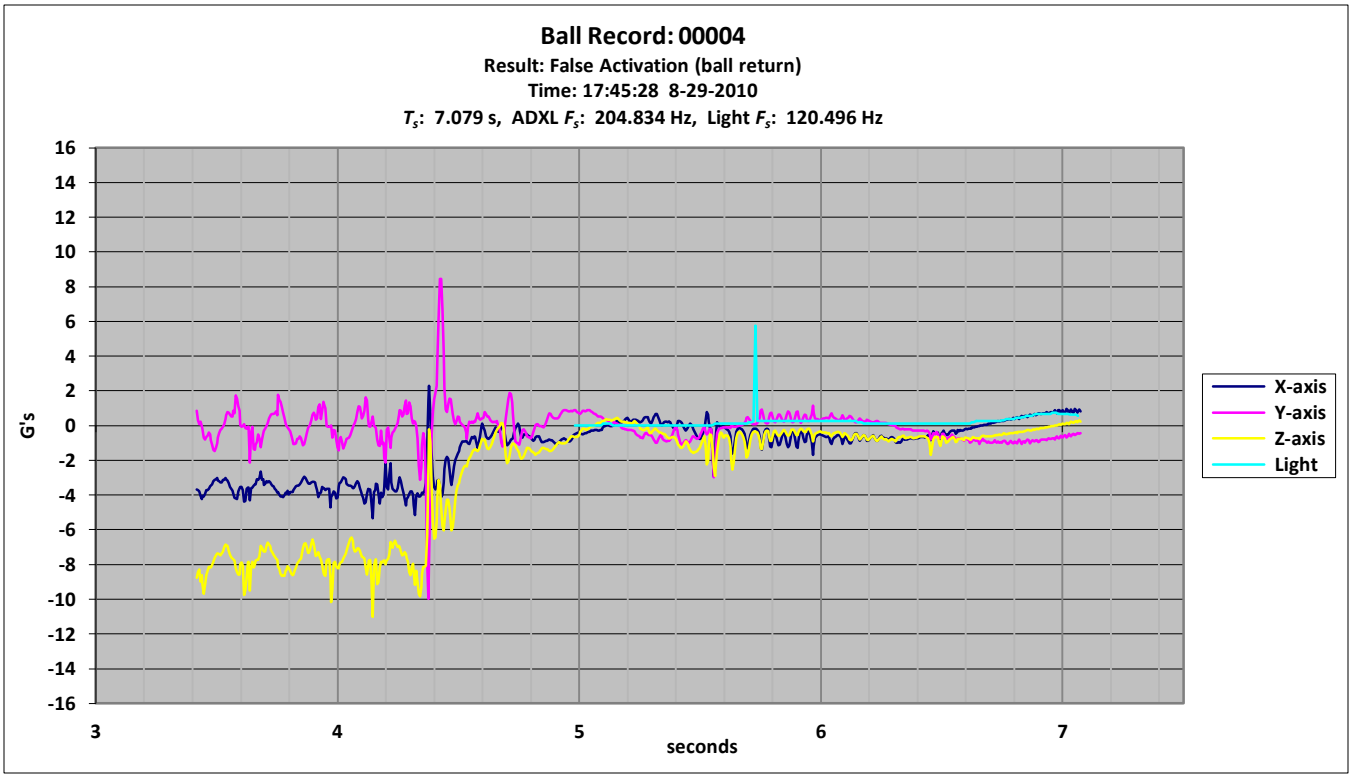

Figure 49: Ball Record 00004 (ball return activation)

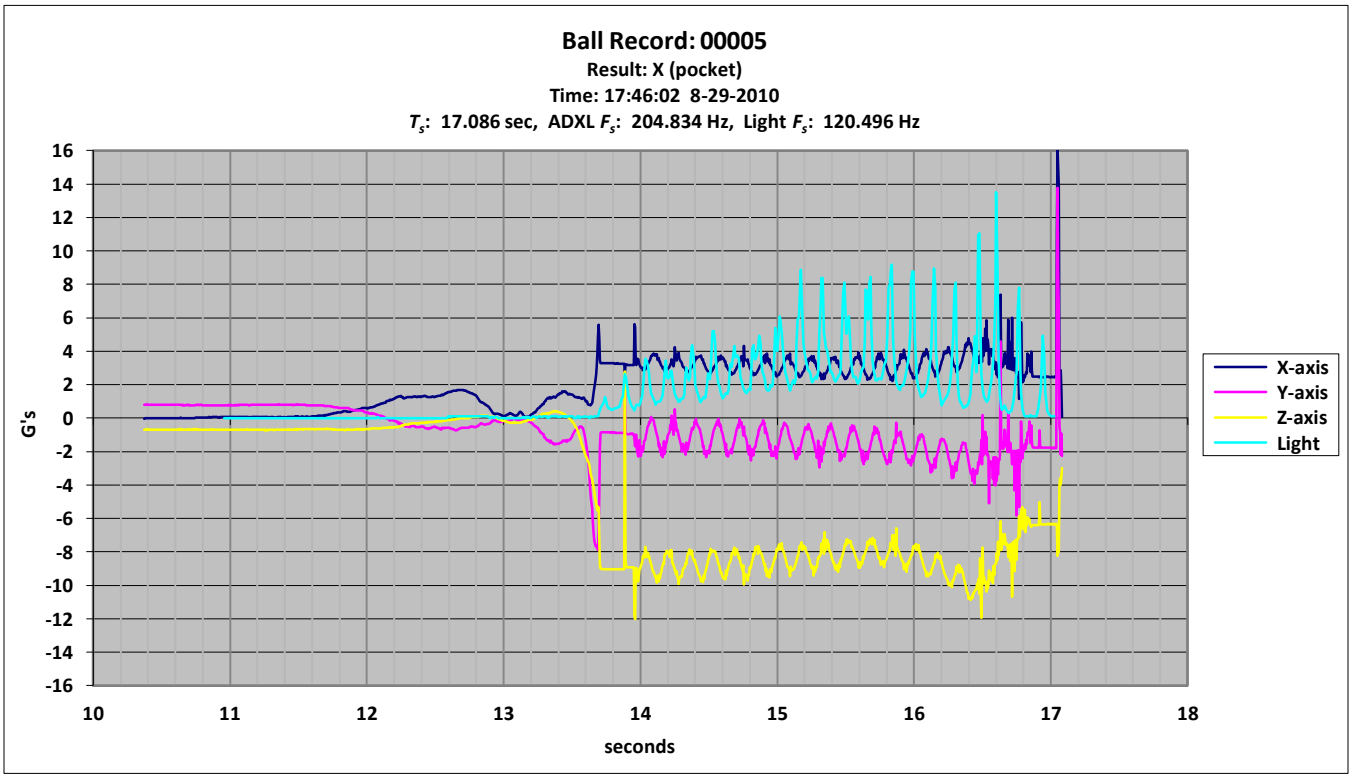

Figure 50: Ball Record 00005 (typical waveform)

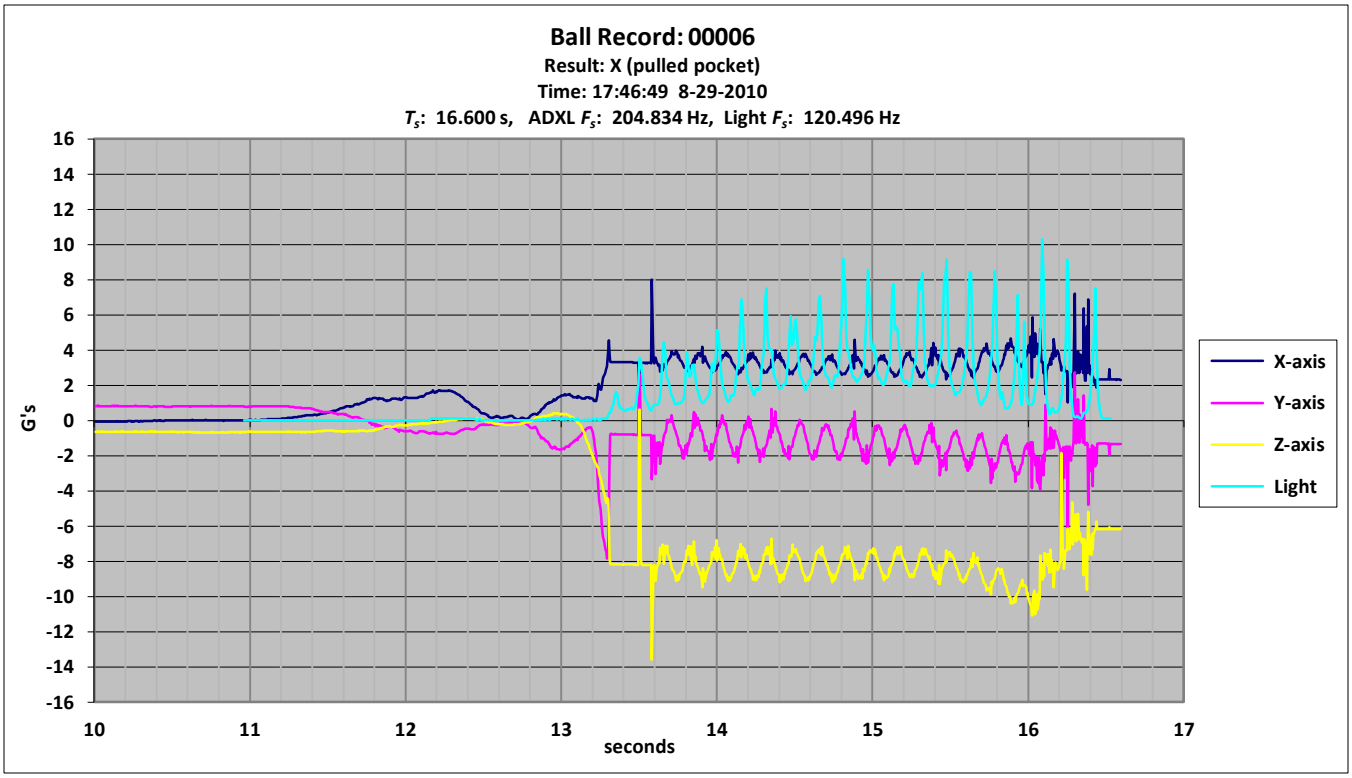

Figure 51: Ball Record 00006 (typical waveform)

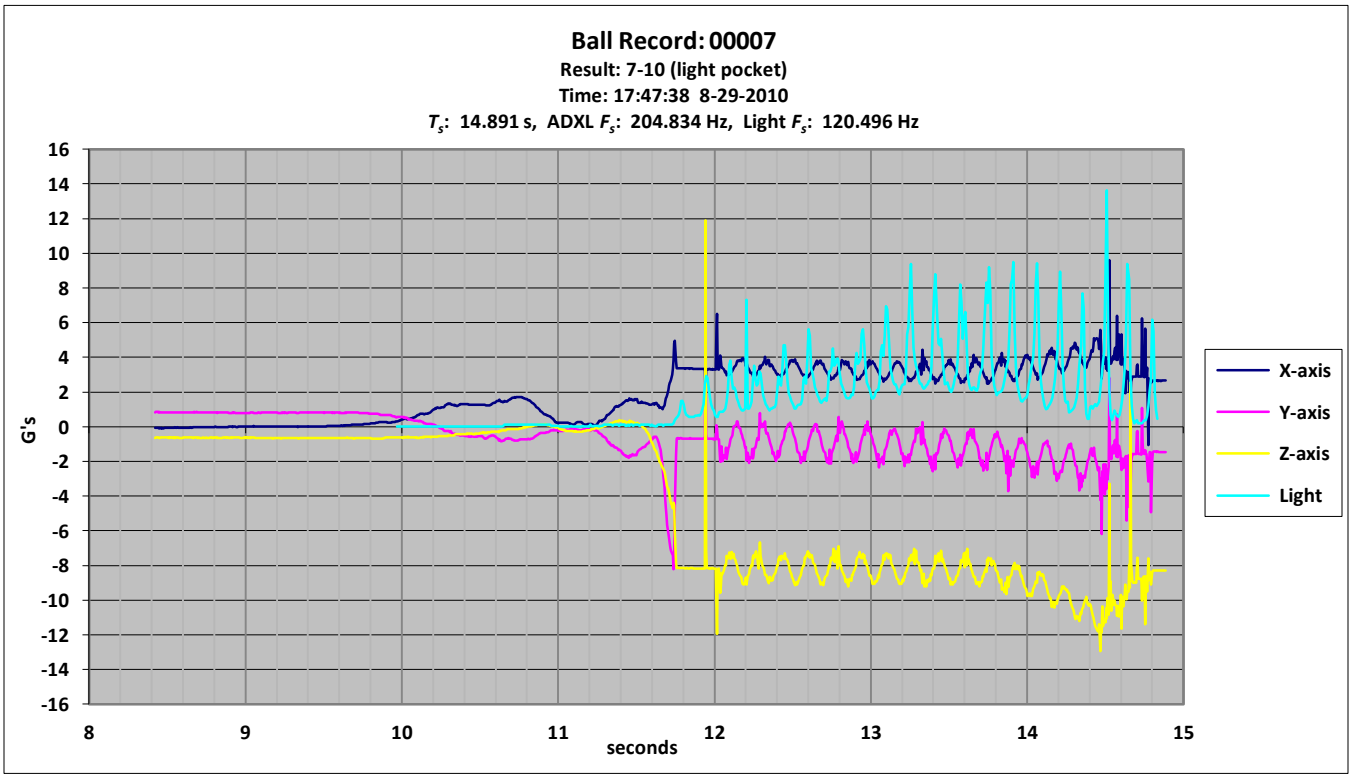

Figure 52: Ball Record 00007 (typical waveform)

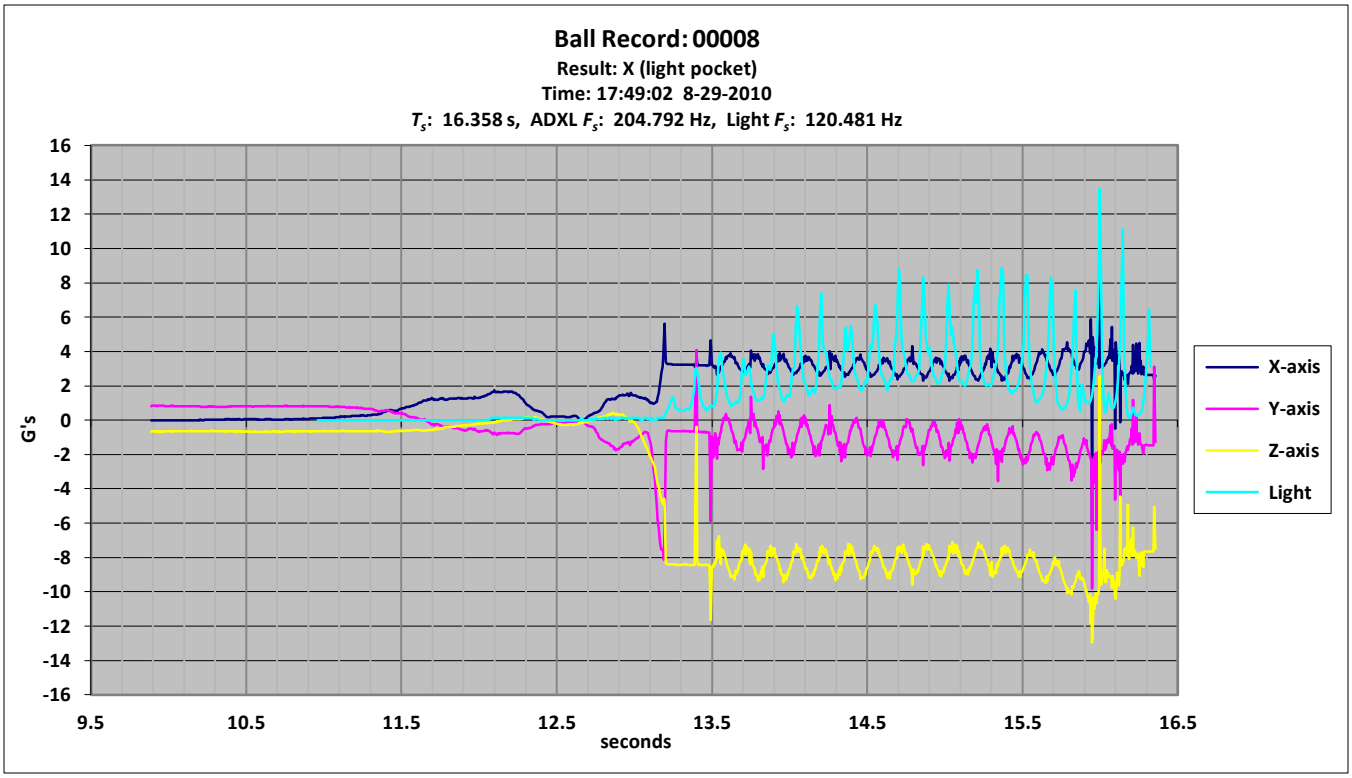

Figure 53: Ball Record 00008 (typical waveform)

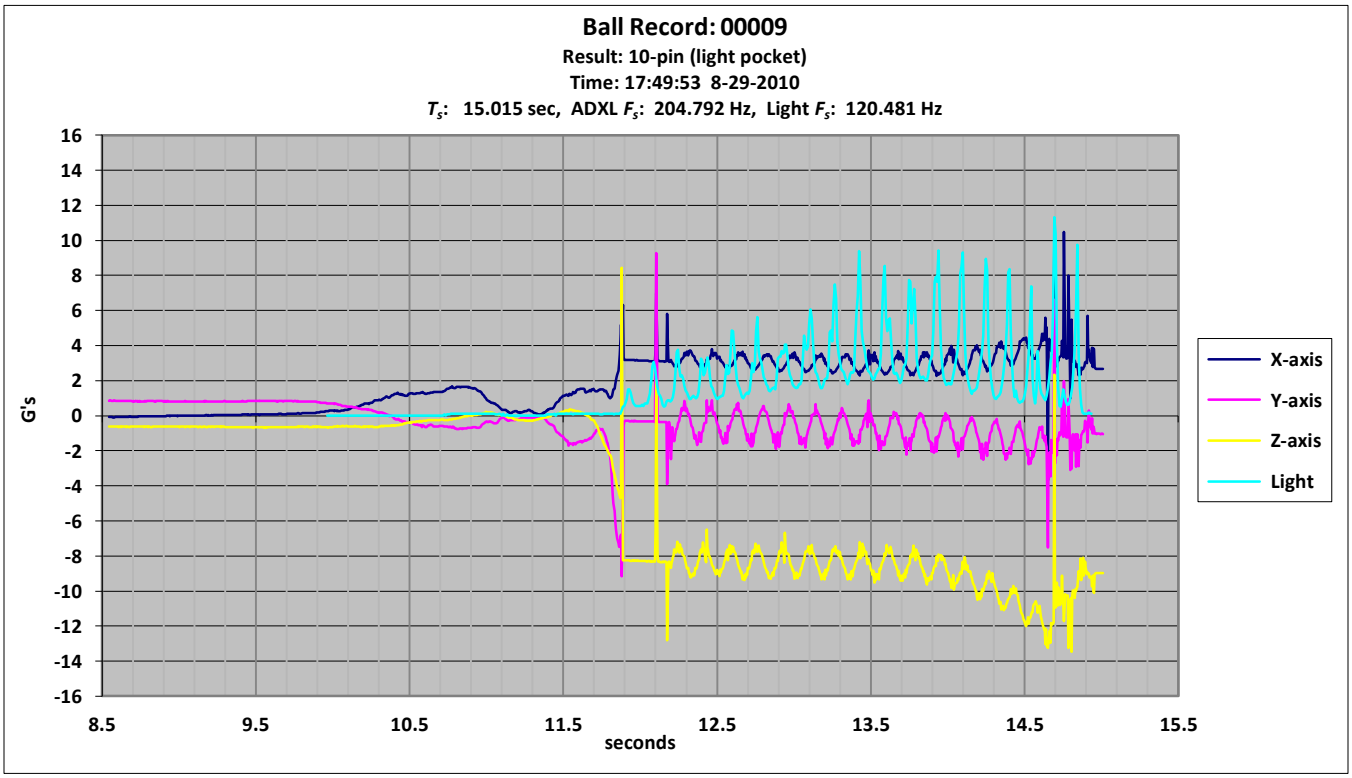

Figure 54: Ball Record 00009 (typical waveform)

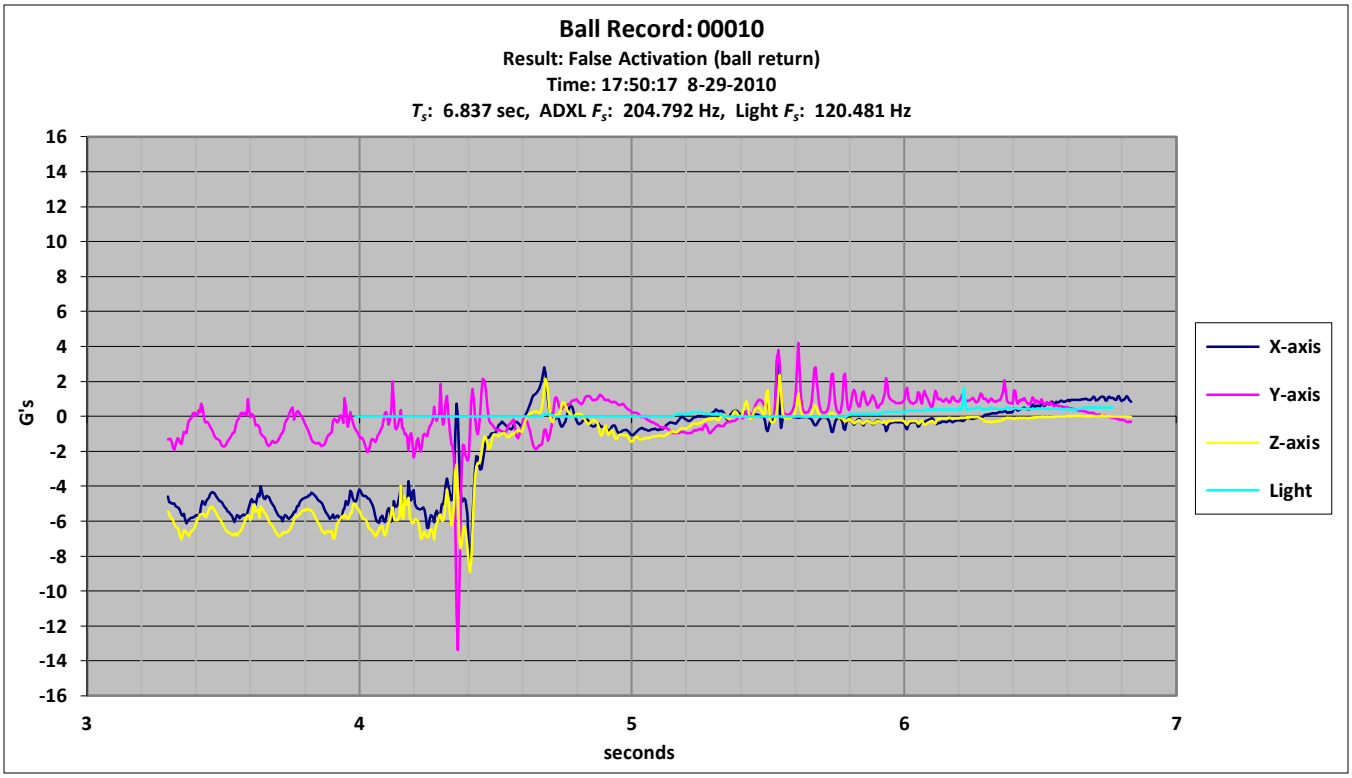

Figure 55: Ball Record 00010 (ball return activation)

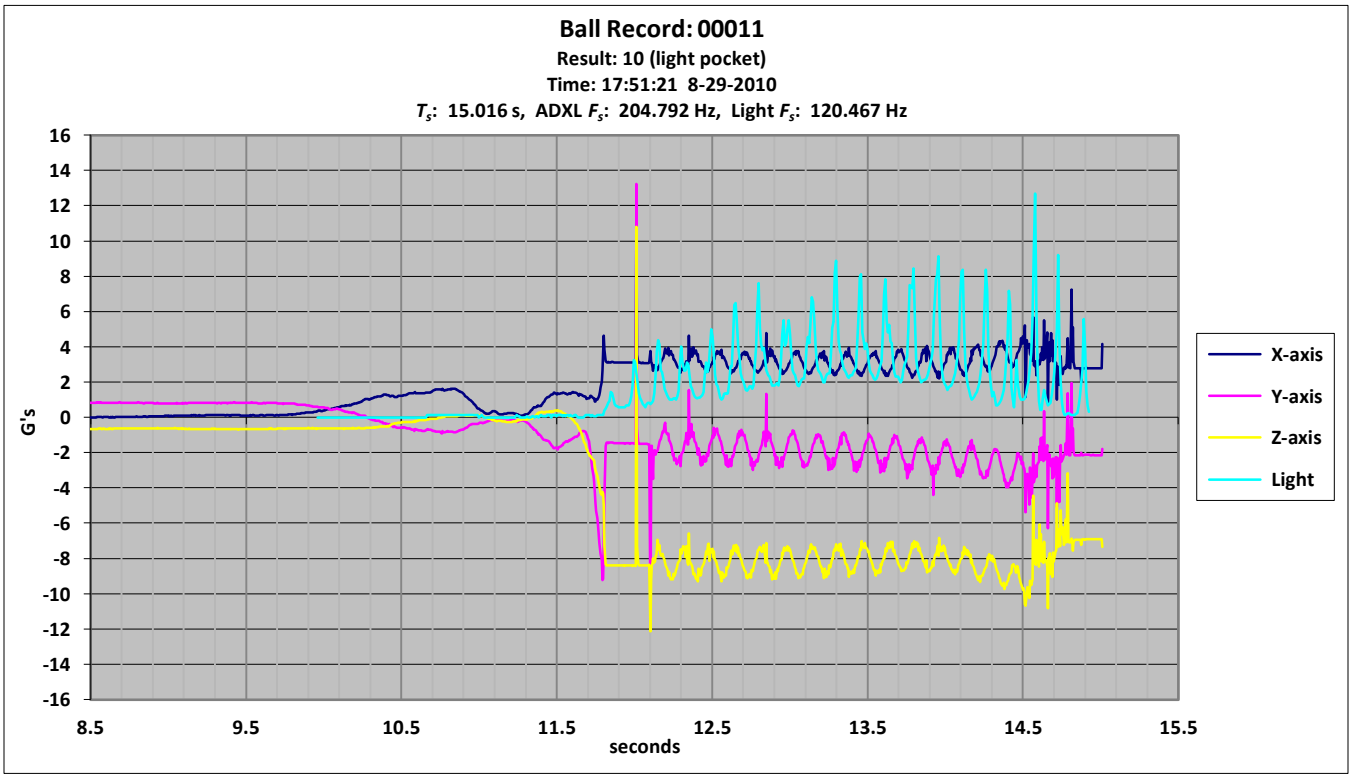

Figure 56: Ball Record 00011 (typical waveform)

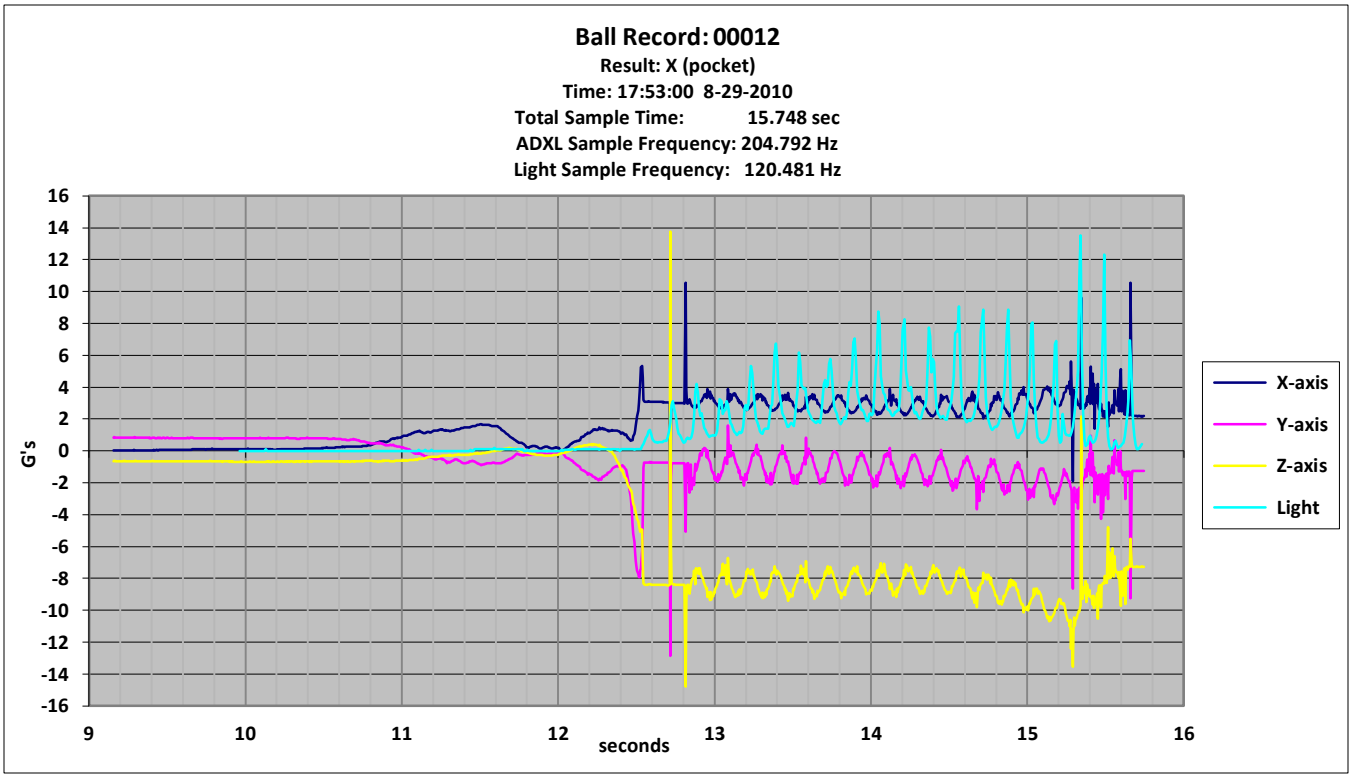

Figure 57: Ball Record 00012 (typical waveform)

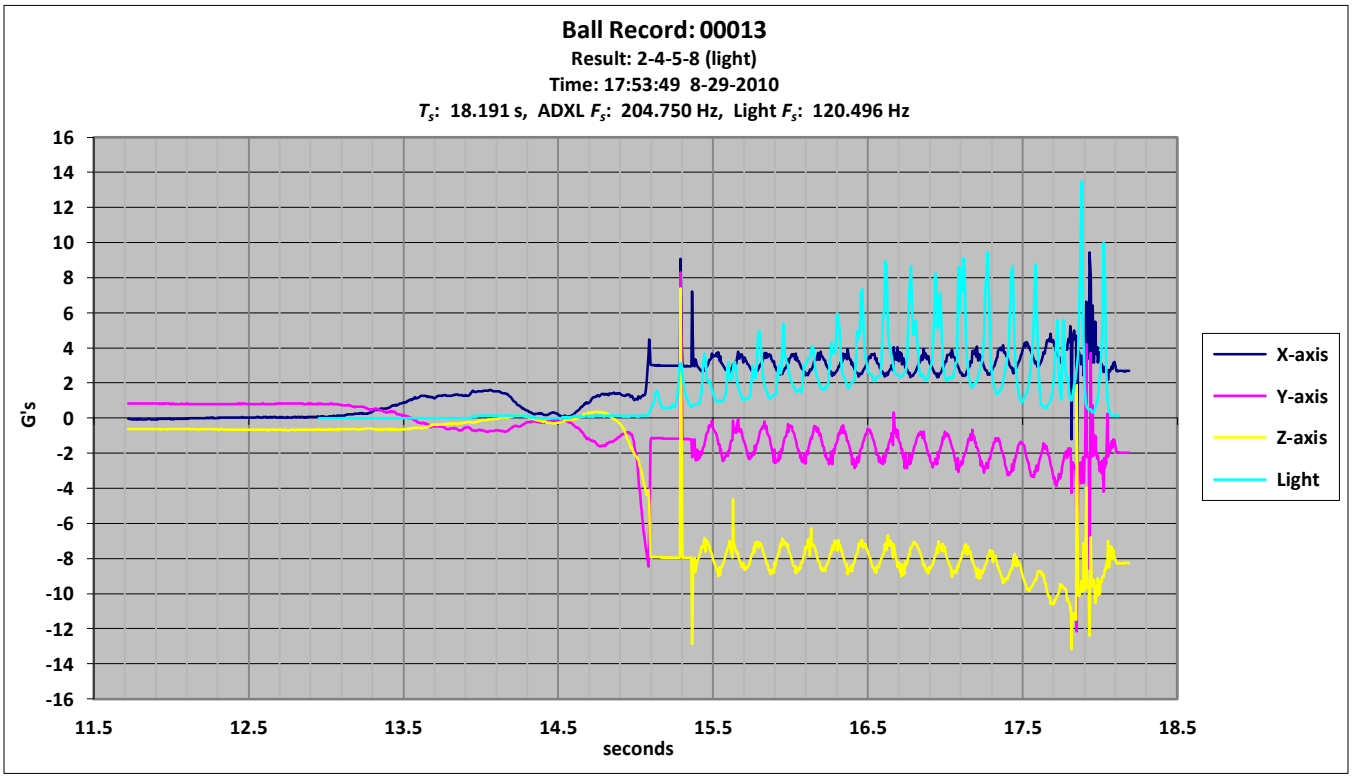

Figure 58: Ball Record 00013 (typical waveform)

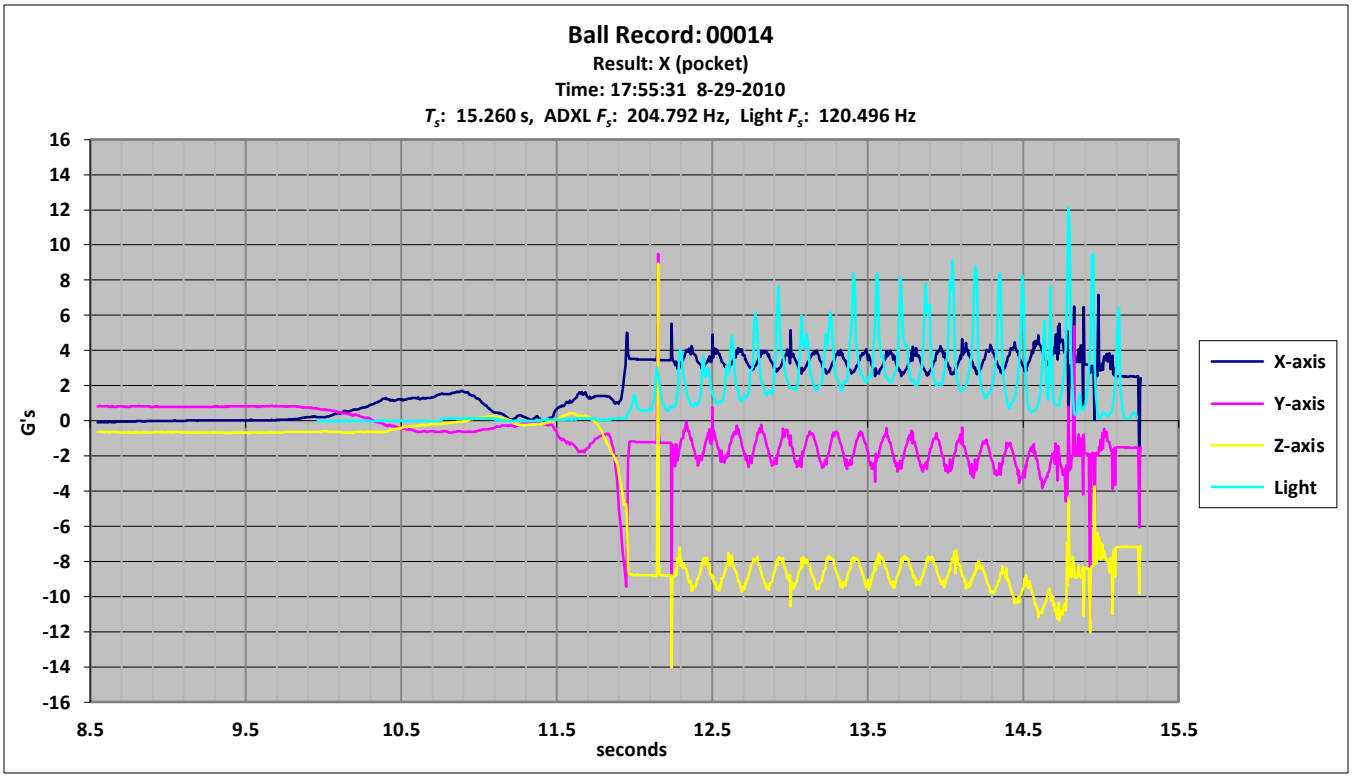

Figure 59: Ball Record 00014 (typical waveform)

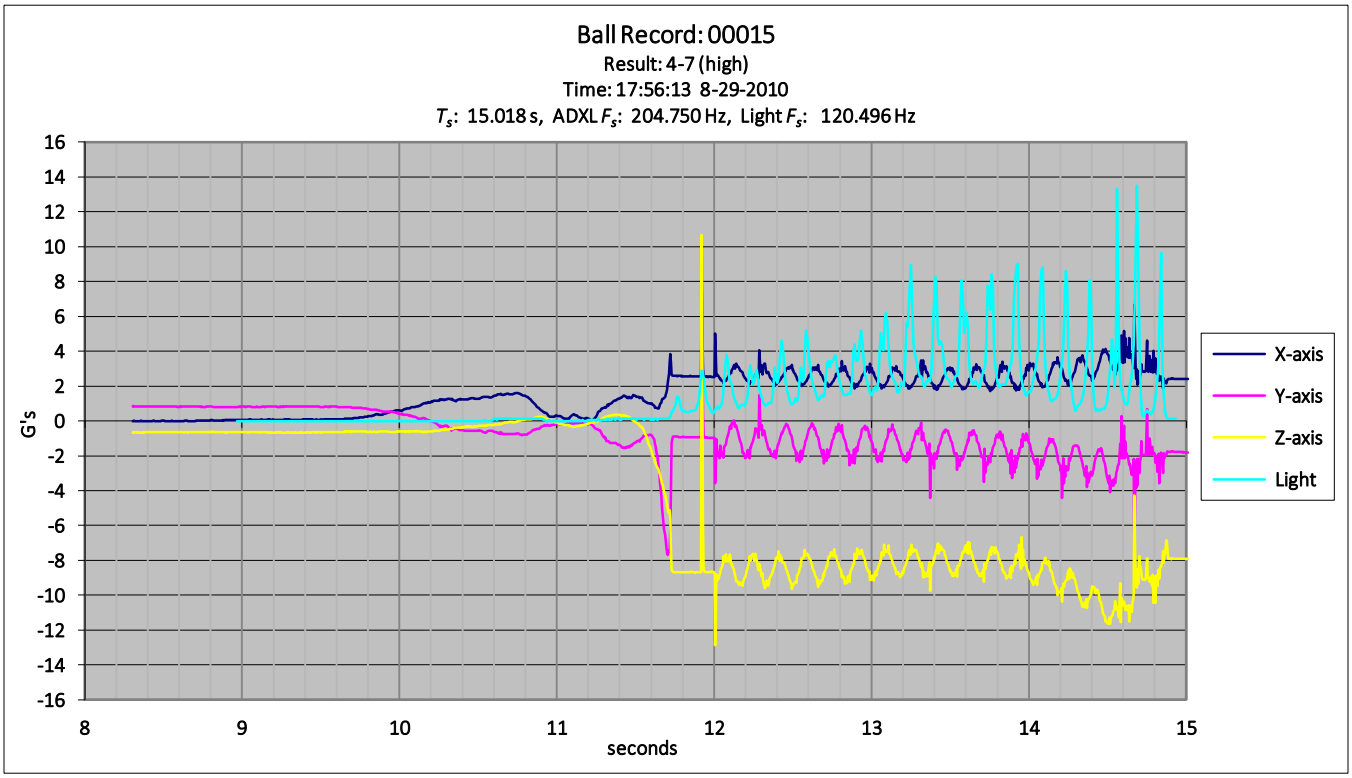

Figure 60: Ball Record 00015 (typical waveform)

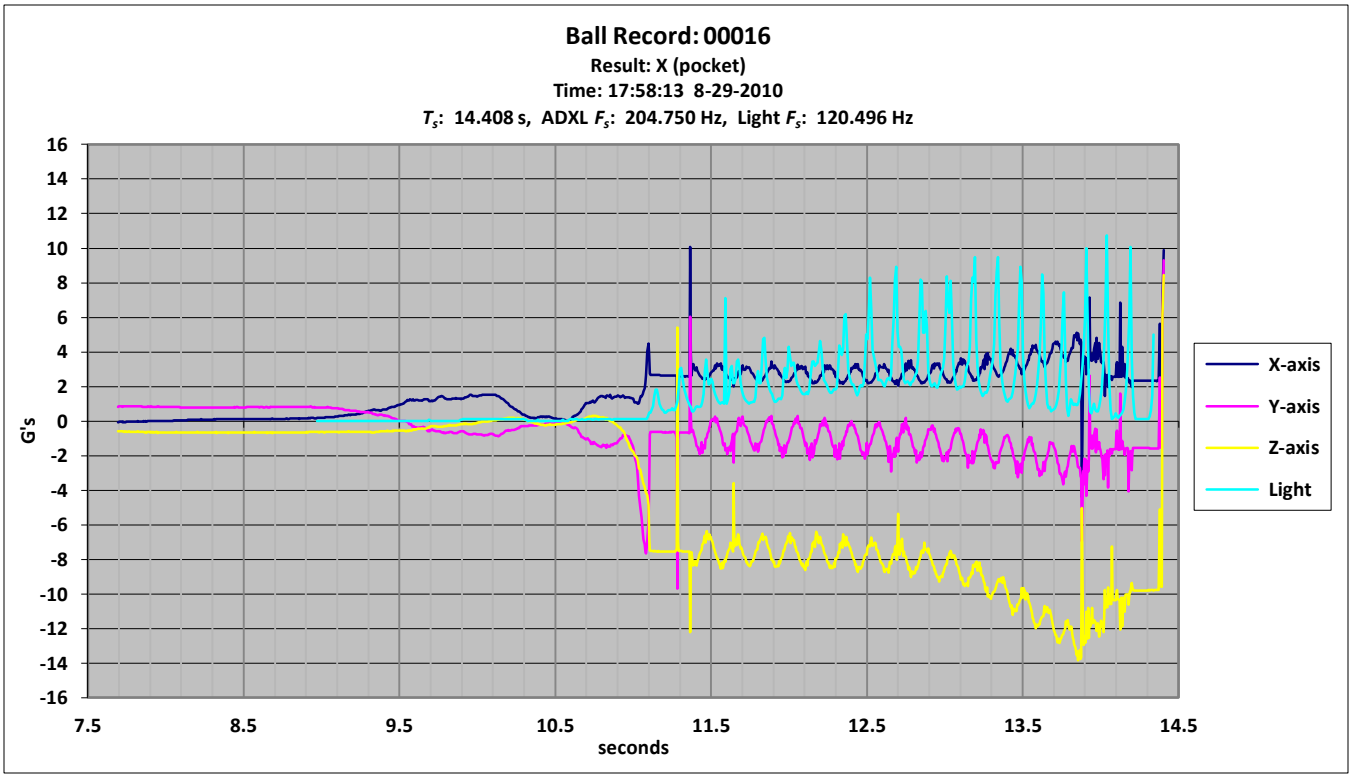

Figure 61: Ball Record 00016 (typical waveform)

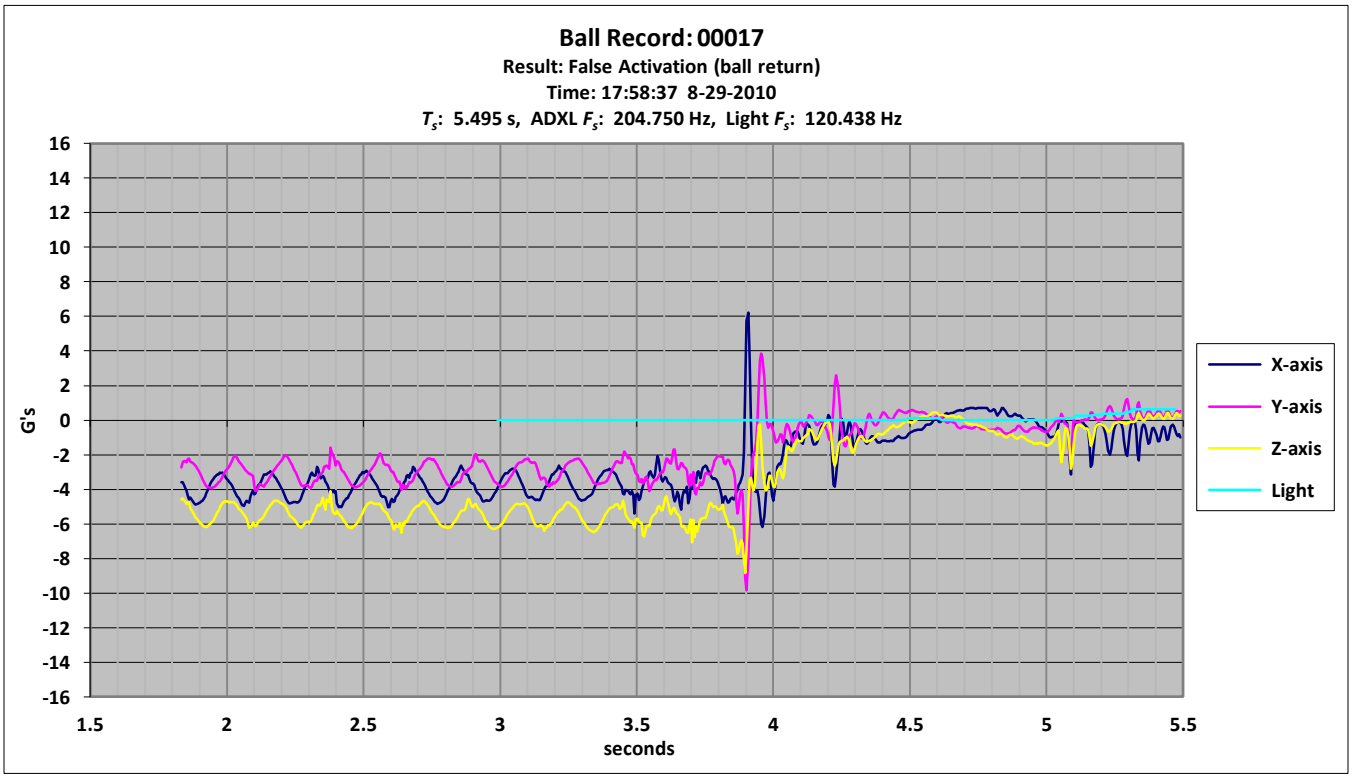

Figure 62: Ball Record 00017 (ball return activation)

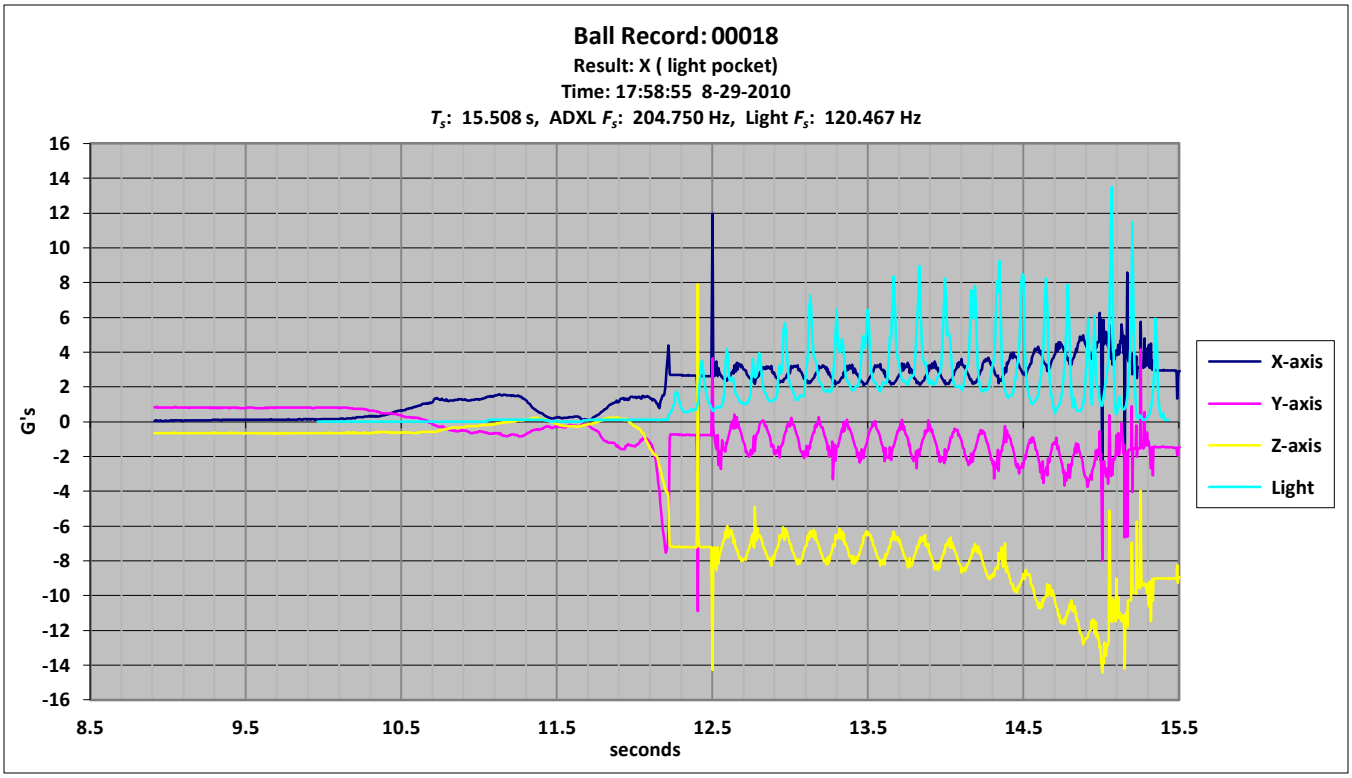

Figure 63: Ball Record 00018 (typical waveform)

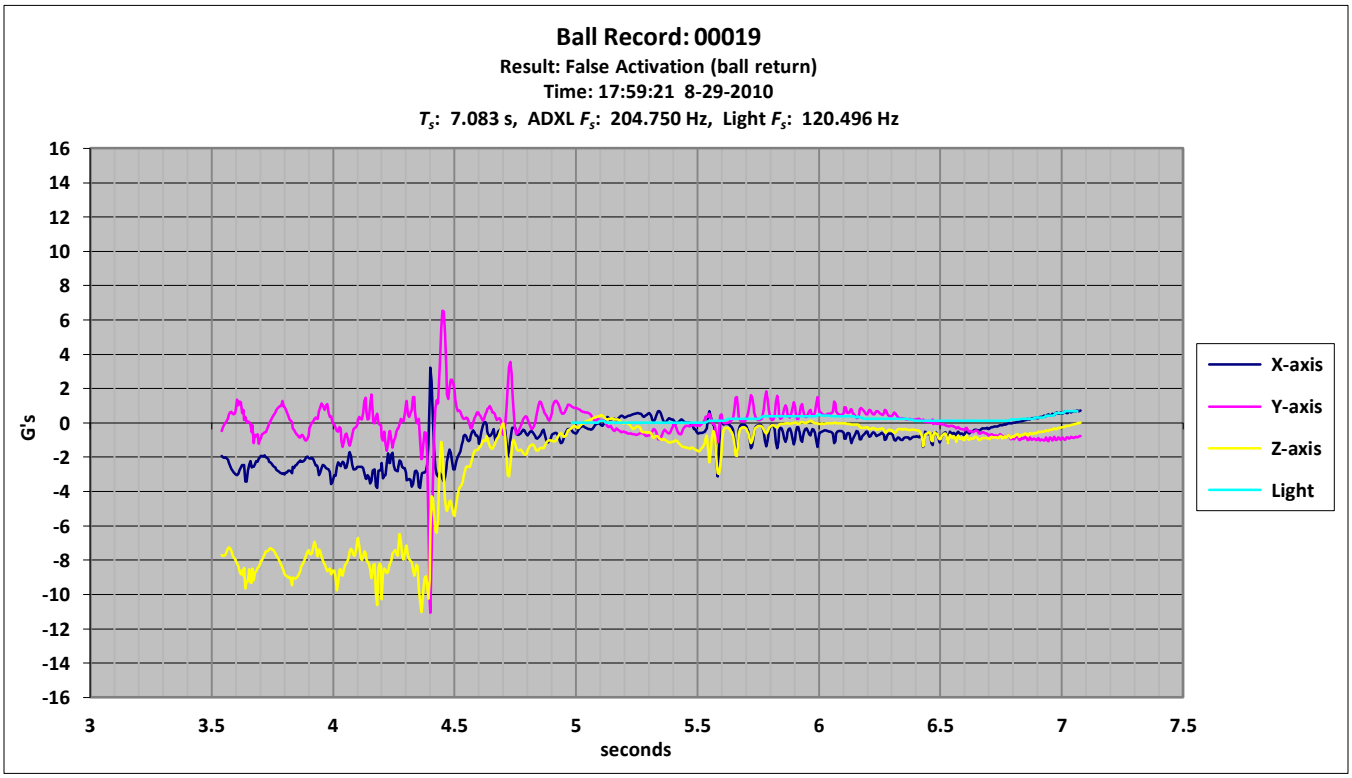

Figure 64: Ball Record 00019 (ball return activation)