

# NEW-GENERATION FULLY PROGRAMMABLE CONTROLLER FOR FUNCTIONAL ELECTRICAL STIMULATION APPLICATIONS

by

Davide Agnello

A thesis submitted in conformity with the requirements for the degree of  
Master of Applied Science  
Graduate Department of the Institute of Biomaterials and Biomedical Engineering  
University of Toronto

# New-Generation Fully Programmable Controller for Functional Electrical Stimulation Applications

Davide Agnello

Master of Applied Science

Graduate Department of the Institute of Biomaterials and Biomedical Engineering  
University of Toronto

2011

## **Abstract**

Functional electrical stimulation (FES) systems have been developed to help restore various neuromuscular functions in individuals with neurological disorders leading to paralysis. Most of the current FES systems are designed for specific neuroprosthesis applications (i.e., walking, grasping, bladder voiding, coughing, etc.) and when one intends to use them in other custom made applications they are very limited due to a lack of functionality and flexibility in hardware and programmability. This prevents effective and efficient development of customized neuroprostheses. Research and development efforts at the Rehabilitation Engineering Laboratory at the University of Toronto were being carried out with an objective to produce a new, fully programmable and portable FES system. This thesis presents a novel proof-of-concept prototype controller for use in the new FES system. The controller subsystem manages and controls the overall FES system including the real-time decoding and execution of stimulation, data acquisition, external systems interfaces and user interface.

## **Acknowledgements**

First of all, my most sincere thanks goes out to my supervisor, Professor Milos R. Popovic, for his guidance and support. His passion towards his work and dedication to his students are truly remarkable.

I would also like to express my gratitude to the members of the Rehabilitation Engineering Laboratory for their support and making this laboratory a great place to work.

Finally, I would like to thank my family, particularly my parents for making this possible with their encouragement and support.

# Table of Contents

Abstract.....	ii
Acknowledgements.....	iii
List of Tables .....	viii
List of Figures.....	xi
List of Acronyms .....	xv
1 Introduction.....	1
1.1 Motivation .....	2
1.2 Thesis Overview.....	3
2 Background.....	5
2.1 Physiological Overview .....	5
2.2 Brief History.....	7
2.3 Stimulation Characteristics .....	8
2.3.1 Stimulation Waveforms .....	8
2.3.2 Stimulation Waveform Parameters.....	11
2.3.3 Stimulation Programmability.....	13
2.4 FES Applications.....	18
2.5 Past and Current FES Systems.....	19
2.5.1 Surface FES Systems .....	19
2.5.2 Implantable/Percutaneous FES Systems.....	25
2.5.3 Summary .....	27
3 Proposed System.....	29
3.1 Complex Motion Stimulator System.....	29

3.2	Concept System.....	30
3.3	Objectives.....	34
4	System Requirements and Analysis.....	36
4.1	System Requirements.....	36
4.1.1	System Requirements and Analysis Tools.....	36
4.1.2	System Context Analysis .....	37
4.1.3	List of Scenarios .....	40
4.1.4	Functional and Non-functional Requirements .....	40
4.1.5	Use Cases .....	44
4.2	System Analysis .....	47
4.2.1	Static System Model .....	48
4.2.2	Dynamic System Models .....	48
5	Hardware and Software Platform Overview.....	50
5.1	Hardware Platform .....	50
5.2	Real-Time Software Platform and Development Toolset .....	57
5.3	System Structure Overview.....	58
6	PC/hand-held Device Interface .....	61
6.1	Communication Protocol Selection.....	61
6.2	Interface Configuration .....	66
6.3	Received Data Processing .....	67
6.4	Stimulator Function Requests .....	68
6.4.1	Bidirectional Stimulation Protocol Transfer.....	69
6.4.2	Stimulation Parameter Change .....	71
6.4.3	Stimulation Parameter Retrieve .....	73

6.4.4	Stimulation Protocol Change .....	74
6.4.5	Stimulator Control through USB Interface .....	75
7	User Interfaces .....	76
7.1	Stimulator Interfaces .....	76
7.1.1	Stimulator Input Interface .....	76
7.1.2	Stimulator Output Interface .....	78
7.2	PC Interfaces .....	81
7.2.1	Stimulation Protocol Interface .....	81
7.2.2	Stimulator Control Interface .....	83
8	Input Sensor Interface .....	87
8.1	Application .....	87
8.2	Types of Sensors .....	88
8.3	Interface Design .....	90
9	Stimulation Protocol .....	94
9.1	Supported Stimulation Tasks & Control .....	94
9.2	Stimulation Protocol Encoding & Packaging .....	101
9.3	Stimulation Protocol Decoding and Execution .....	107
9.3.1	Overall decoding and execution design structure .....	107
9.3.2	Primitive decoding and execution .....	111
10	Experimental Results .....	137
10.1	Experimental Setup .....	137
10.2	PC Interface Testing .....	141
10.3	Stimulation Protocol Decoding and Execution Testing .....	146
10.3.1	Stimulation Protocol Function Testing .....	146

10.3.2	Stimulation Protocol Timing Testing.....	163
11	Conclusion & Future Work.....	167
11.1	Major Accomplishments for First FES Prototype .....	167
11.2	Areas of Extension and Future Work .....	168
	References.....	171
	Appendix 1: Requirements Models .....	176
	Appendix 2: Static System Model .....	183
	Appendix 3: Dynamic System Models .....	184
	Appendix 4: Stimulation Protocol Encoding.....	200
	Appendix 5: Experimental Results Timing Data.....	211

## List of Tables

Table 2-1: Application specific stimulation frequency.....	12
Table 2-2: Stimulator systems technical specifications.....	28
Table 4-1: List of interface entities (users, devices, and other subsystems).....	39
Table 4-2: Sample list of scenarios.....	40
Table 4-3: Functional Requirements.....	43
Table 4-4: Non-Functional Requirements .....	43
Table 4-5: Use case start stimulation.....	45
Table 4-6: Download stimulation protocol.....	46
Table 5-1: Controller hardware platform minimum requirements .....	51
Table 5-2: Summary of advantages and disadvantages of microcontroller architectures .....	52
Table 5-3: Summary of advantages and disadvantages of FPGA architectures .....	52
Table 6-1: PC/hand-held device interface functionality .....	62
Table 6-2: PC/hand-held device to stimulator interface general function requirements .....	62
Table 6-3: Summary of common communication standards applicable to design .....	63
Table 6-4: Wireless communication standard comparison.....	64
Table 6-5: Wired communication standard comparison.....	65
Table 6-6: USB operation code functions.....	68
Table 6-7: Parameter change identification bits .....	72
Table 7-1: Comprehensive list of status messages managed by the controller subsystem.....	80
Table 7-2: Comprehensive list of error messages managed by the controller subsystem .....	81
Table 8-1: Summary of applicable sensors with applications. ....	89
Table 9-1: Summary of supported stimulation primitives. ....	99



Table 9-2: Block specific data in encoded stimulation protocol.....	103
Table 9-3: Primitive byte encoding summary.....	104
Table 9-4: Primitive byte encoding.....	106
Table 9-5: No-stimulation primitive encoding .....	114
Table 9-6: Ramp up PW execution index values example. ....	117
Table 9-7: Amplitude change primitive byte encoding. ....	119
Table 9-8: Frequency change primitive byte encoding.....	120
Table 9-9: ‘Marker A’ & ‘Jump A’ primitive encoding.....	122
Table 9-10: Text primitive encoding .....	123
Table 9-11: Stimulation attribute randomization primitives.....	126
Table 10-1: Stimulator/PC communication test timing summary. ....	144
Table 10-2: Global default stimulation parameters .....	147
Table 10-3: Channel specific default stimulation parameters.....	148
Table 10-4: Randomization parameter summary.....	155
Table A1-1: Complete list of fundamental scenarios .....	176
Table A1-2: Download stimulation protocol use case.....	177
Table A1-3: Upload stimulation protocol use case.....	177
Table A1-4: Start stimulation use case .....	178
Table A1-5: Stop stimulation use case.....	178
Table A1-6: Pause stimulation use case.....	179
Table A1-7: Resume stimulation use case .....	179
Table A1-8: Adjust stimulation parameters use case.....	180
Table A1-9: Adjust stimulation protocol use case.....	180
Table A1-10: Process stimulation protocol use case .....	181

Table A1-11: Capture data use case.....	181
Table A1-12: Sensor or electrode failure use case.....	182
Table A1-13: Battery charge in progress use case.....	182
Table A1-14: Battery fully charged use case.....	182
Table A4-1: Stimulation protocol byte encoding details. ....	205
Table A4-2: Primitive encoding details. ....	210

## List of Figures

Figure 2-1: Illustration of direct motor neuron stimulation (adopted from [2]).	6
Figure 2-2: Neuroprosthesis performing palmer grasp and release (adopted from [29]).	6
Figure 2-3: Stimulation waveform types (adopted from [30], where A. is a DC, B. is an AC and C. is a pulse shaped stimulation waveforms).	9
Figure 2-4: Pulse-shaped stimulation waveform configurations	10
Figure 2-5: Variation of amplitude and PW required for muscle tissue excitation (adopted from [30]).	13
Figure 2-6: ON and OFF stimulation pattern illustration (adopted from [30]).	14
Figure 2-7: Stimulation ramp up and down illustration (adopted from [30]).	15
Figure 2-8: Amplitude change illustration (adopted from [30])	15
Figure 2-9: Pulse randomization illustration.	16
Figure 2-10: Bionic Glove demonstration	20
Figure 2-11: NESS H200 grasping neuroprostheses	20
Figure 2-12: ODFS drop foot stimulator system	21
Figure 2-13: NESS L300 drop foot neuroprostheses.	21
Figure 2-14: WalkAide on leg illustration	22
Figure 2-15: Parastep system	22
Figure 2-16: EMPi 300PV Stimulator	23
Figure 2-17: MOTIONSTIM 8 stimulator system.	23
Figure 2-18: Compex-2 programmable FES system	24
Figure 2-19: Compex-3 programmable FES system	24
Figure 2-20: Freehand system component overview	25

Figure 2-21: NeuRX DPST™ RA/4 diaphragm pacing system.....	26
Figure 3-1: Compex Motion Stimulator System.....	30
Figure 3-2: High level diagram of the new generation FES system .....	31
Figure 3-3: New generation FES system component diagram overview .....	32
Figure 3-4: New generation FES system functional diagram overview .....	33
Figure 4-1: Control subsystem context diagram.....	38
Figure 4-2: Controller subsystem use case diagram .....	47
Figure 4-3: Start stimulation (user) sequence diagram .....	49
Figure 5-1: Block diagram of the LPC2368 microcontroller (adapted from [48]) .....	56
Figure 5-2: MCB2300 evaluation board with LPC2368 microcontroller (adapted from [49]) ...	57
Figure 5-3: Clock generation diagram for the LPC2300 series (adapted from [51]).....	59
Figure 6-1: RS-232 serial sniffer schematic (Adopted from [58]) .....	70
Figure 6-2: Bit composition of the parameter change identification byte .....	72
Figure 7-1: External interrupt 0 switch input diagram (adopted from [60]).....	78
Figure 7-2: LCD display interface to LPC2368 (adopted from [60]).....	79
Figure 7-3: Program Compex Motion primary window .....	82
Figure 7-4: Stimulator's control center PC software .....	85
Figure 8-1: Sample hardware de-bounce circuitry (adopted from [63]).....	91
Figure 8-2: Before proper hardware debouncing is performed (adopted from [63]).....	92
Figure 8-3: Potentiometer sensor interface diagram to LPC2368 microcontroller (adopted from [60]).....	93
Figure 9-1: Compex Motion main programming window.....	96
Figure 9-2: Hand grasping stimulation protocol.....	100
Figure 9-3: Example stimulation protocol to illustrate primitive encoding.....	105

Figure 9-4: High-level stimulation decoding and execution flow chart. ....	108
Figure 9-5: Process next parameters block flow chart.....	111
Figure 9-6: Stimulation channel 1 programmable ramps. ....	116
Figure 9-7: Programmable amplitudes with transition times.....	118
Figure 9-8: Jump primitive loop example.....	121
Figure 9-9: DAC to speaker interface diagram on MCB 2300 evaluation board (adopted from [60]).....	125
Figure 9-10: PDF for uniform random number generator .....	127
Figure 9-11: Synchronization example on a two channel stimulator .....	129
Figure 9-12: Synchronization example solution.....	130
Figure 9-13: User interaction simple trigger example. ....	133
Figure 9-14: User interaction complex trigger example.....	134
Figure 9-15: User branch primitive example.....	136
Figure 9-16: User branch A trigger criteria .....	136
Figure 10-1: High level experimental setup block diagram.....	139
Figure 10-2: Stimulation protocol data used for transfer testing. ....	143
Figure 10-3: PW configurations for ramps A and B.....	149
Figure 10-4: stimProtTest01 stimulation protocol.....	150
Figure 10-5: stimProtTest01 stimulation results captured from Matlab program. ....	151
Figure 10-6: stimProtTest02 stimulation protocol.....	152
Figure 10-7: stimProtTest02 stimulation results.....	153
Figure 10-8: stimProtTest03 stimulation protocol.....	154
Figure 10-9: stimProtTest03 stimulation results.....	156
Figure 10-10: stimProtTest04 stimulation protocol.....	157

Figure 10-11: stimProtTest04 stimulation results.....	158
Figure 10-12: stimProtTest05 stimulation protocol.....	159
Figure 10-13: Branch Primitive Triggers.....	160
Figure 10-14: User interaction G trigger signal.....	161
Figure 10-15: stimProtTest05 stimulation results.....	163
Figure A2-1: Static System Model – UML Class Diagram.....	183
Figure A3-1: Download Stimulation Protocol Sequence Diagram.....	184
Figure A3-2: Upload Stimulation Protocol Sequence Diagram .....	185
Figure A3-3: Start Stimulation (PC/hand-held Device) Sequence Diagram .....	186
Figure A3-4: Start Stimulation (user) Sequence Diagram.....	187
Figure A3-5: Stop Stimulation (PC/hand-held Device) Sequence Diagram .....	188
Figure A3-6: Stop Stimulation (user) Sequence Diagram .....	189
Figure A3-7: Pause Stimulation Sequence Diagram .....	190
Figure A3-8: Resume Stimulation Sequence Diagram.....	191
Figure A3-9: Adjust Stimulation Parameters Sequence Diagram .....	192
Figure A3-10: Adjust Stimulation Protocol Sequence Diagram.....	193
Figure A3-11: Process Stimulation Protocol Sequence Diagram.....	194
Figure A3-12: Capture Data Sequence Diagram .....	195
Figure A3-13: Electrode Failure Sequence Diagram.....	196
Figure A3-14: Sensor Failure Sequence Diagram .....	197
Figure A3-15: Battery Charge in Progress Sequence Diagram .....	198
Figure A3-16: Battery Fully Charged Sequence Diagram.....	199

## List of Acronyms

Acronym	Description
FES	Functional Electrical Stimulation
SCI	Spinal Cord Injury
AP	Action Potentials
DC	Direct Current
AC	Alternating Current
TENS	Transcutaneous Electrical Nerve Stimulation
NMES	Neuromuscular Electrical Stimulation
ODFS	Odstock Dropped Foot Stimulator
FDA	U.S. Food and Drug Administration
EMG	Electromyography
USB	Universal Serial Bus
DPS	Diaphragm Pacing System
PC	Personal Computer
GUI	Graphical User Interface
HCP	Health Care Provider
UML	Unified Modeling Language
TTL	Transistor-Transistor Logic
RISC	Reduced Instruction Set Computer
ARM	Advanced RISC Machine
AVR	Advanced Virtual RISC
PIC	Peripheral Interface Controller
FPGA	Field Programmable Gate Array
UART	Universal Asynchronous Receiver Transmitters
RTOS	Real-Time Operating System
CPU	Central Processing Unit
RS-232	Recommended Standard 232
CDC	Communications Device Class
EINT	External Interrupt
ISR	Interrupt Service Routine
LCD	Liquid Chrystal Display
PW	Pulse-Width
API	Application Programming Interface
COM Port	Communication Port
RC	Resistor-capacitor
IRQ	Interrupt Request
HEX	Hexadecimal
TMR	Timer
PB	Push Button
LED	Light Emitting Diode

# 1 Introduction

The loss of voluntary function in limbs and other body parts, also known as paralysis, is caused by disease or injury to the neuromuscular system. A subset of such disorders results from injury or disease that affects the central nervous system. Typical examples which lead to these disorders are stroke, spinal cord injury and traumatic brain injury. A common outcome of these injuries, which are classified as “central” since they occurred in the central nervous system, is that the lower motoneurons, which innervate muscles, remain intact. Thus, if electrical stimulation is applied to these neurons they can generate muscle contractions, which are otherwise paralyzed due to injury or disease to the central nervous system [1].

Since the early 1960’s electrical stimulation has been used to artificially generate body functions such as walking, grasping, and hearing [2]. Various commercially available electronic stimulators have been developed to restore these functions [3-13]. The majority of them have been developed having one application in mind and thus have limited flexibility, i.e., cannot be applied to restore other body functions [2]. In recent years, it has been suggested that the creation of a single programmable, portable and multipurpose electrical stimulator would allow practitioners to apply electrical stimulation to diverse patients in various applications. One of the long term objectives of our team was to develop a portable and fully programmable electrical stimulator that can be used in various functional electrical stimulation (FES) applications. FES is the use of electrical stimulation to generate or restore specific body functions.



The focus of my work was to conduct the appropriate research in relevant FES systems and applications, and use these results to develop the proof-of-concept controller subsystem that will be an integral part of the proposed FES system. This component manages and controls the execution of stimulation in the stimulator. The controller subsystem would also allow one to program any stimulation sequence or protocol, and allow the use of various external input signals to trigger and/or control the stimulation. Furthermore, this component would enable one to program, control and receive feedback from the stimulator using a personal computer (PC) or handheld device. The proposed electrical stimulator will be designed such that practitioners will be able to develop various custom-made neuroprostheses, neurological assessment devices, electrical stimulation systems and experimental setups for electro-physiological studies using the same hardware platform.

## **1.1 Motivation**

According to the Heart and Stroke Foundation [14], stroke was the third leading cause of death in Canada in 2006. Each year more than 50,000 strokes occur in Canada and after the age of 55 the risk of stroke doubles every 10 years. As of 2009, there were approximately 300,000 Canadians living with the effects of stroke. Out of 100 people who have suffered a stroke, 15 die, 10 recover completely, 25 recover with minor impairment or disability, 40 are left with moderate to severe impairment, and the last 10 are so severely disabled that they require long-term care. Stroke costs the Canadian economy \$3.6 billion a year (2000 statistic).

According to ICORD (International Collaboration on Repair Discoveries) [15], there were approximately 40,000 individuals living in Canada with Spinal Cord Injury (SCI) in 2009. Each year approximately 1,050 Canadians sustained SCI with an incident rate of about 35

persons/million. Financial care requirements over the lifetime of an individual with SCI can vary from \$1.25 million for a low thoracic paraplegic to \$5 million for a high cervical quadriplegic (2009 statistics).

FES therapy has the potential to assist individuals who are moderately to severely affected by stroke or SCI in restoring voluntary motor functions. This rehabilitation technique has shown great promise. A considerable number of studies [1, 16-28] indicate that the use of FES as a form of rehabilitation therapy provides the ability to restore voluntary body functions in individuals suffering from paralysis. For example, results obtained from [18] in improving voluntary hand and arm functions. Such results were attained from weekly training programs spanning 2 to 3 months pending patients' pathology and availability to participate in the training.

## **1.2 Thesis Overview**

This thesis describes a controller subsystem for use in the novel FES system being developed at the Rehabilitation Engineering Laboratory (REL) at the University of Toronto and is organized as follows.

Chapter 2 goes over the required background research conducted in the area of FES neuroprostheses, including stimulation characteristics, relevant applications and previously developed FES systems.

Chapter 3 presents the high-level vision of the new FES system, including information on the overall system concept and the objectives for the controller subsystem.

Chapter 4 provides a system requirements break down including an analysis overview for the desired controller subsystem, based on the research conducted in the background section.

Chapter 5 gives an overview of the high-level system design decisions on which the rest of the project is based on; including decisions on hardware platform, real-time development software and structure.

Chapters 6 to 9 concentrate on the core research, design and implementation of the various components, which make up the controller subsystem.

Chapter 10 presents the experimental setup and results which have been used to verify and validate the system's design and operation.

Chapter 11 summarizes the accomplishments, and possible areas of extension and future work.

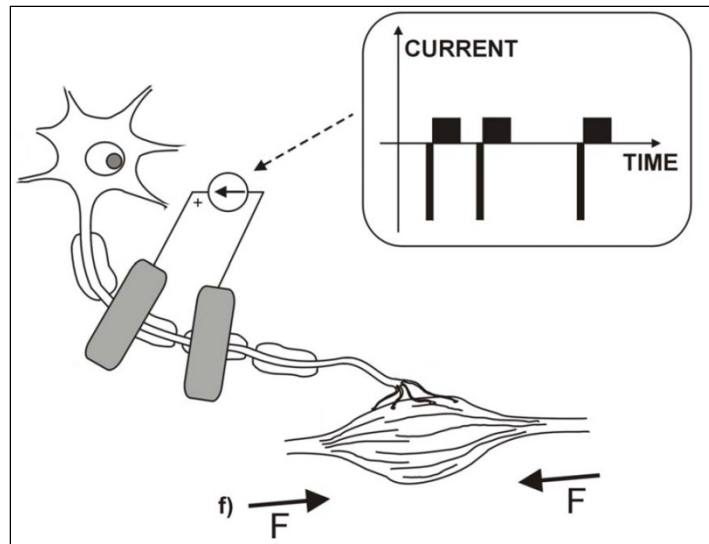
## 2 Background

This chapter provides the required background information pertaining to FES systems, starting with a physiological overview of electrical stimulation and brief history of neuroprostheses. Then, the research results associated to the present state of the art in the FES field including potential applications and operating ranges of existing FES devices are presented.

### 2.1 Physiological Overview

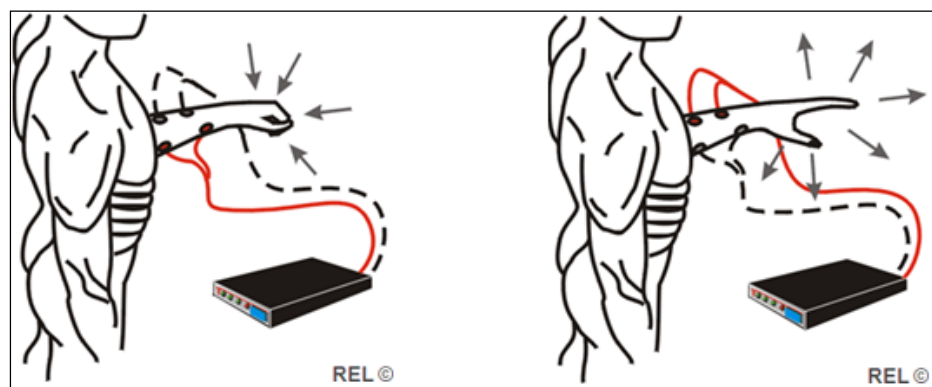
Information in nerve cells is coded and transmitted as series of electrical impulses called action potentials (AP), which represent brief changes in cell electric potential of about 80 mV. APs can be artificially generated by inducing electric charge into the nerve cell or nerve axon. The intensity of the signal transmitted is directly proportional to the frequency of APs that occur in the axon per unit of time. When APs are generated using electrical stimulation and are used to produce a body function, it is referred to as FES. During FES, for every AP that propagates towards the end of the axon that is innervating a muscle (orthodromic propagation) one AP will propagate backwards towards the cell body of the motoneuron (antidromic propagation) [2]. FES is typically concerned with orthodromic propagation as they generate muscle contractions in order to produce the desirable body function. Figure 2-1 below illustrates the direct stimulation of a motoneuron which then innervates the specific muscle. In the case when the APs are generated by the central nervous system (instead of FES), the cell body receives AP driven inputs from dendrites, it summates the excitatory and inhibitory APs, processes them and decides whether or not to generate an output AP. Following stroke or SCI the motoneurons do not receive appropriate input from the central nervous system therefore inhibiting muscle function.

FES replaces this functionality by artificially generating required AP's to elicit a desired muscle/limb function.



**Figure 2-1: Illustration of direct motor neuron stimulation (adopted from [2]).**

A neuroprosthesis is an FES device that provides short bursts of electrical impulses to the central or peripheral nervous system in order to generate sensory and/or motor function as depicted in Figure 2-2 below.



**Figure 2-2: Neuroprosthesis performing palmar grasp and release (adopted from [29]).**

Some example neuroprostheses, which have been developed in the past 40 years, include cochlear implants, bladder management stimulator systems, walking and grasping neuroprostheses [2]. Common components found in each neuroprostheses include a power source, a controller, a stimulus generator, user interface and electrodes. Nerves can be stimulated using surface, percutaneous or implanted electrodes. The stimulation waveforms are realized through current or voltage pulses which can be monophasic or biphasic. Monophasic pulses which deliver a single mono-polar pulse at a time are not usually used as they lead to an unbalanced charge delivery to the tissue, which can cause tissue damage [30]. Most modern FES systems implement biphasic current or voltage pulses, where the first pulse evokes an action potential and the following pulse removes the delivered charge, thereby preventing tissue damage from accumulated charge in the surrounding tissue.

## **2.2 Brief History**

The first “electric stimulators” may be dated back to 46 A.D. by Scribonius Largus, a pharmacist from Neron and physician to the Roman Emperor Claudius who applied electrical currents from the torpedo fish to treat various ailments such as headaches and painful gout [31]. The breakthrough in the medical use of electricity came from the invention of the electrostatic generator (~1663) and the discovery of the Leyden jar (1745) – the origin of the capacitor [31-32]. In 1766 Johann Gottlieb Sc  ffer published a book which provided a detailed description of the use of electrical discharges for treatments associated with paralysis and pain. According to Sc  ffer, the greatest therapeutic effect was obtained with “shaking electricity” – when sparks were used to transmit the electricity to the patient, inducing strong muscular twitches.

In the early 1960s, Liberson and colleagues developed a simple neuroprostheses intended to correct drop foot. Drop foot is characterized by a lack of dorsiflexion during the swing phase of gait, resulting in short shuffling strides [33]. Liberson's device is known as the first neuroprostheses which received a patent and was the initial push towards FES systems and devices. Liberson's neuroprosthesis is comprised of a power supply worn on a belt, two surface electrodes to stimulate the common peroneal nerve, and a heel switch. The stimulation was activated whenever the heel rose off the ground and deactivated when the heel contacts the ground.

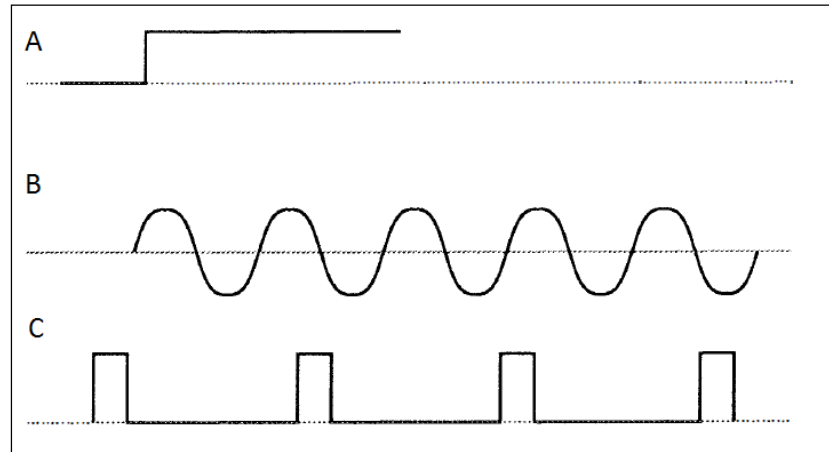
## **2.3 Stimulation Characteristics**

FES systems use electrical signals to activate nerve fibers and muscles in order to perform the desired function, these signals can be characterized in terms of shape, frequency, pulse-width (PW) and amplitude. Controlling stimulation parameters as a function of time, temporal activation of individual channels, and overall stimulation patterns are key aspects in the development of custom neuroprostheses.

### **2.3.1 Stimulation Waveforms**

Throughout the years various stimulation waveforms have been used to provide neural excitation. These stimulation waveforms, summarized in Figure 2-3 below, can be grouped into direct current (DC) flow (A), alternating current (AC) flow (B) or pulse shaped current flow (C). DC waveforms have been used clinically to increase circulation, to treat neuralgia, for electrolysis and as a tool for iontophoresis (transfer of ions through the skin) [30]. This type of

stimulation waveform does not provide the means to activate muscle contractions; rather it produces muscle twitches associated with the start and end of the DC waveform.



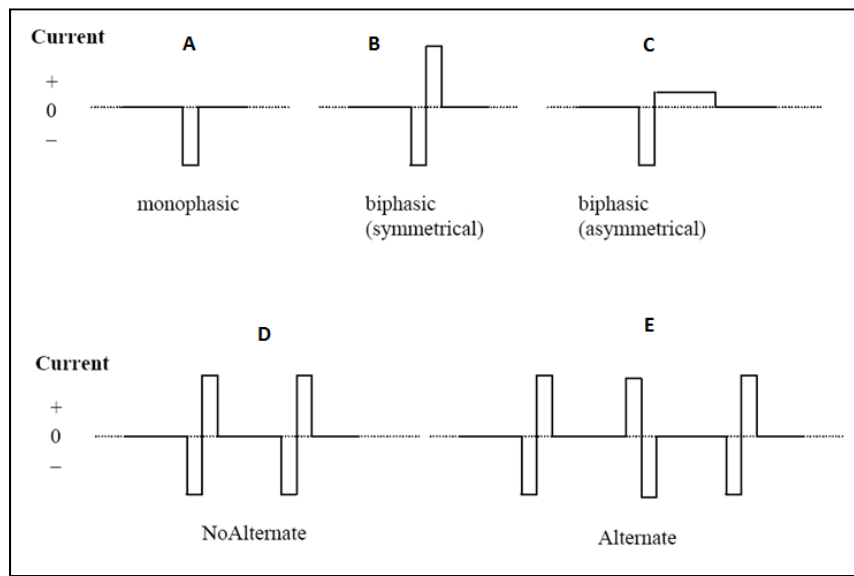
**Figure 2-3: Stimulation waveform types (adopted from [30], where A. is a DC, B. is an AC and C. is a pulse shaped stimulation waveforms)**

AC waveforms are defined as a constant current flow moving in one direction and then in the reverse direction. These waveforms can be of any shape including sinusoidal, square, triangular, and trapezoidal. Common characteristic of such waveforms are the lack of electrical silence between phases. Similar to DC waveforms, AC waveforms are not usually used for therapeutic stimulation, unless they are packaged into on-off envelopes (with electrical silence between each envelope) or further modulation. Low frequency sinusoidal stimulation is known to be effective in activating denervated muscles [34].

Pulse shaped waveforms have contributed to the effectiveness of today's practical FES applications such as transcutaneous electrical nerve stimulation (TENS) for pain management and neuromuscular electrical stimulation (NMES) programs for muscle strengthening. These pulsed current waveforms can be defined as providing discrete electrical pulses of known waveform, frequency, amplitude and PW. Most of today's FES is done using pulsed stimulation



waveforms. Pulse shapes commonly used are rectangular, which rise abruptly (eliminating any concern of nerve accommodation), stays at constant amplitude for a determined period of time and then falls abruptly.



**Figure 2-4: Pulse-shaped stimulation waveform configurations**

These rectangular pulses can be generated in different configurations as shown in Figure 2-4 above, each having its own purpose and use. Monophasic pulses (Figure 2-4 – A) move current only in one direction. When these pulses are used for TENS or NMES applications, they have the likelihood of causing electrode deterioration and tissue damage (skin irritation or rash when surface stimulation is used) when used for prolonged periods of time (over an hour) [30]. This effect is due to the altering of ionic distributions and causing polarization which leads to tissue breakdown and burns. Although the effect of ionic distribution is not desirable, monophasic waveforms are still used in some short-term therapeutic TENS applications.

The unequal ion transfer can be reduced by using biphasic (symmetrical or asymmetrical) stimulation pulses. In the case of asymmetric biphasic waveforms (Figure 2-4 – C) one direction

of current is enough to cause excitable tissue to depolarize, while the opposite direction is lower in amplitude but proportionally longer in duration minimizing neural excitation. The overall effect of the stimulation is similar to the Monophasic waveform, but with a reduction of ion redistribution. The area under these pulses indicates the amount of charge that is delivered to the tissue. These waveforms can be either balanced or unbalanced – in terms of the area under each pulse within the period. The most common and desirable biphasic waveform is the balanced charge pulse waveform, which further reduces issues of ion build-up. The symmetric biphasic waveform shown in Figure 2-4 – B has equal amplitude and duration in both directions. Since both the positive and negative pulses are equal, they both are effective in causing depolarization in the excitable tissue and are usually used for the activation of large muscles [30]. Overall most FES applications desire the use of biphasic pulse waveforms, especially when they are used for prolonged periods of time.

Another characteristic used in generating a pulsed waveform in FES applications is that of alternating the positive and negative pulses after each period as seen in Figure 2-4 – E. This waveform pattern is used in conjunction with each type of pulse waveform described above (Figure 2-4 – A to C). These pulses are used for customized FES applications that are developed to enhance performance in professional athletes.

### **2.3.2 Stimulation Waveform Parameters**

As described in Section 2.3.1, electrical stimulation can be delivered through various types of waveforms which work best in certain applications. In modern FES systems, the waveforms most commonly used are pulse shaped waveforms. Aside from the configuration of overall pulse

pattern, these waveforms all have a pulse frequency, amplitude and PW which can be configured to elicit the desired functionality.

### 2.3.2.1 Frequency

The frequency of the stimulus signal directly influences the strength and quality of the evoked motor response [30]. Therefore the pulse repetition is directly related to the rate at which neural fibers produce AP's. Over the years, specific stimulation frequencies ranges have been determined to work best depending on the intended application, these frequency-range values based on the electrical stimulation application have been summarized in Table 2-1 below.

Application	Frequency Range (Hz)
Neuromuscular	20 - 50
Spasm reduction and relaxation	1-3
Increased blood flow	7-9
Endurance	10-20
Resistance and hypertrophy	50-70
Strength	75-100
Explosive strength	101-120

**Table 2-1: Application specific stimulation frequency**

### 2.3.2.2 Amplitude and PW

The amplitude and PW of the stimulus signal must be intense enough to at least meet the threshold of the excitable tissue to generate AP's. In today's FES systems typical values for PW are between 50 and 800  $\mu$ s, if the PW is lower, higher amplitudes are required to achieve the required depolarization threshold. Stimulators which provide both amplitude and PW control are more commonly found in units designed for pain management, but occasionally these controls

are also found in neuromuscular stimulation systems as well [30]. Figure 2-5 below displays the amplitude vs. PW relation for the threshold limit for excitation to near maximal muscle contraction and motor response. This data was collected from stimulating the wrist extensors using surface electrodes, with a frequency of 32 Hz and a constant-current stimulator.

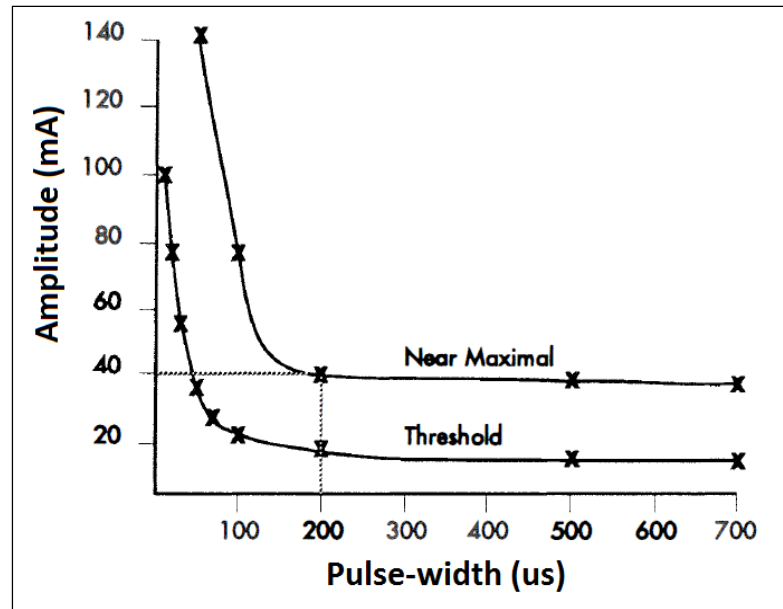


Figure 2-5: Variation of amplitude and PW required for muscle tissue excitation (adopted from [30]).

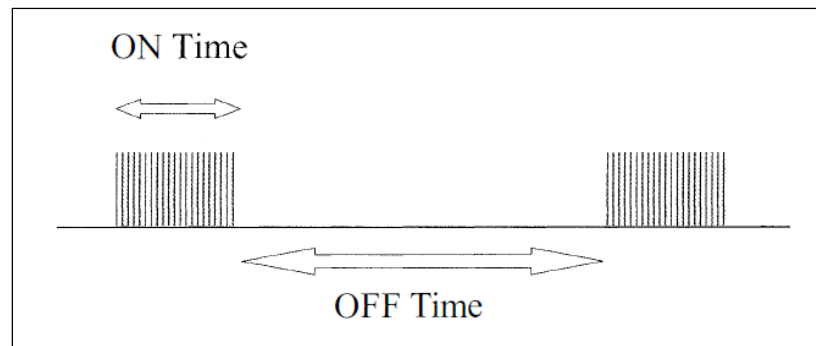
### 2.3.3 Stimulation Programmability

The degree of programmability that a universal stimulator must be capable of on top of configurable waveforms and parameters is especially important in neuromuscular stimulators intended to perform complicated functional tasks such as grasping, reaching, walking, and standing. This added degree of complexity introduces more capacity and intelligence within the stimulator to provide the user or clinician with the ability to configure and program the stimulator system to perform the desired function(s). Descriptions of stimulation

programmability functionality will be provided with possible example applications. The initial assumption is that the stimulator is able to provide constant stimulation with the desired initial waveform and parameters on each of its channels.

### 2.3.3.1 No stimulation

During a stimulation sequence having the ability to stop stimulation on a particular channel for a certain period of time is an important requirement. This capability is essential when trying to use a multiple channel stimulator to stimulate specific muscles at different times or if one channel is used to provide more than one distinct muscle contraction on a single muscle with a pause between each. One can utilize this ‘OFF’ stimulation in series with ‘ON’ stimulation intervals to achieve these kinds of stimulation patterns as shown in Figure 2-6 below.

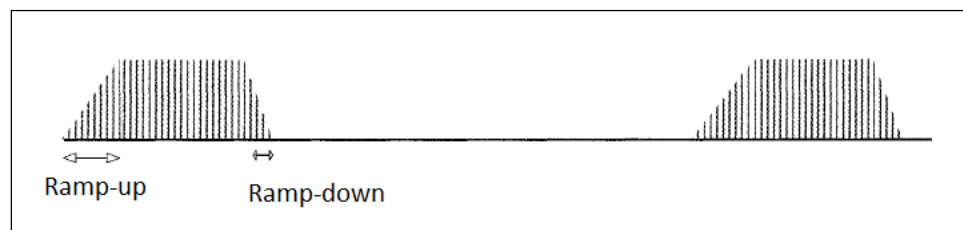


**Figure 2-6: ON and OFF stimulation pattern illustration (adopted from [30]).**

### 2.3.3.2 Stimulation ramps

The comfort of stimulation can be increased with the use of ramps. Ramps gradually increase or decrease the intensity of stimulation (PW or amplitude) over a pre-defined period of time, as shown in Figure 2-7 below (in this particular figure amplitude intensity control was used). This feature provides gradual excitation of increasing/decreasing numbers of nerve fibers that are

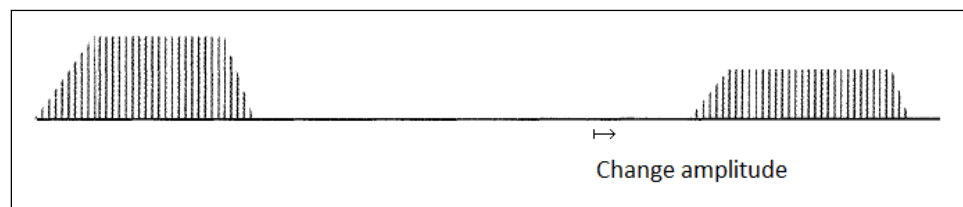
activated using FES. There is no difference in sensation or quality of nerve fiber recruitment between increasing amplitude or PW variables, most stimulators today use PW modulation to create the ramp effect [30]. Ramps can be used to ramp up or ramp down the stimulation intensity. Ramp up is commonly used to provide comfort for the individual receiving the stimulation. Ramp down is used mostly as a safety precaution, for example in allowing a limb to return to resting position in a controlled manner.



**Figure 2-7: Stimulation ramp up and down illustration (adopted from [30]).**

### 2.3.3.3 Adjustable waveform parameters

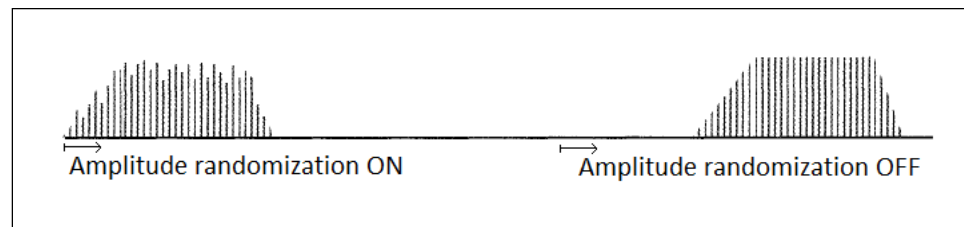
The ability to pre-configure change of stimulation waveform parameters during a stimulation sequence is another valuable function. These pre-configured parameter changes can be associated with changing the stimulation frequency, amplitude and PW. This type of stimulus pattern is used when producing cycling limb functions, such as walking, where during one part of the movement cycle one level of muscle contraction is required and during another part of the movement cycle a different level of muscle contraction is needed.



**Figure 2-8: Amplitude change illustration (adopted from [30])**

### 2.3.3.4 Waveform parameter randomization

In some cases it is desired to randomize one or more stimulation waveform parameter such as stimulation frequency, amplitude and or PW. The ability to randomize one or more waveform characteristic has been shown to reduce muscle fatigue and adaption to stimulation. For example, the randomization of PW has been shown to significantly decrease muscle fatigue in standing and walking by complete thoracic-level paraplegics [35]. Figure 2-9 below displays enabling amplitude-randomization followed by disabling the randomization.



**Figure 2-9: Pulse randomization illustration**

### 2.3.3.5 Stimulation channel synchronization

When the stimulation protocol makes use of multiple stimulation channels, there usually arises a need to synchronize the stimulation between these channels, i.e., to temporarily coordinate the stimulation channels. The ability to enable channel synchronization at specific points in the protocol is a frequent requirement, for example in neuroprostheses for grasping and walking, where certain muscles need to be activated in a coordinated manner.

### 2.3.3.6 Task loops

It is frequently required to create FES protocols in which certain stimulation patterns are repeated a certain number of times. These protocols may have a stimulation pattern nested

within another pattern. In such cases, nested loops of stimulation patterns need to be created and for that purpose nested loop functionality is required. In some cases these nested loops can be infinite in nature, i.e., they are occurring continuously as long as the stimulator is active.

### **2.3.3.7 User interactions**

User interactions are used to help activate/trigger the stimulator. For that purpose one would use a function that continuously monitors digital and/or analog inputs. When a specified analog or digital input pattern is detected by the stimulator, the stimulator uses that signal to trigger the stimulation or to cause a change in the stimulator's state [36]. For example, analog inputs (such as a sliding potentiometer) can be integrated to regulate intensity of stimulation, i.e., as the analog signal increases in voltage, the stimulation amplitude increases [25]. The user interactions are critical as they allow the user to initiate, pause and re-direct the stimulation protocol.

### **2.3.3.8 User alerts (visual/audible)**

The final stimulation characteristic in which can be advantageous to integrate as part of a stimulation protocol is the use of visual and or audible alerts. The stimulator can produce a text message and/or specific sound to alert the user in the current state of the stimulator, i.e., to indicate important changes in the stimulation protocol that is delivered by the FES system. Some examples of such events include safety stimulator shut-down warnings and indications that the stimulator will start or resume stimulation.



## 2.4 FES Applications

Today electrical stimulation is used in various applications: a) for total or partial function substitution (for example neuroprostheses for walking, grasping, and bladder voiding). In these applications electrical stimulation is better known as functional electrical stimulation (FES). b) For reducing the level of motor impairment or for improving the psychiatric condition of the patient through neuromodulation (deep brain stimulation in patients with Parkinson's disease, essential tremor and depression), and c) for restoring voluntary motor function following short-term use of FES as a therapy (improving voluntary walking, reaching and grasping functions). The devices used in these applications deliver the desired electrical stimulation through surface, implanted or percutaneous electrodes. This thesis is concerned with FES applications which include both orthotic and therapeutic applications (application areas a) and c) detailed above).

Surface FES systems apply self-adhesive or non-adhesive electrodes placed on the skin surface just above the muscle that needs to be stimulated. These FES systems are external to the body and used as orthoses to provide function substitution or as therapeutic systems to help restore voluntary motor functions following a neurological injury or diseases [3-11]. Implanted FES systems are intended for more permanent applications, i.e., orthoses that are used to substitute a function at all times [12]. Most, if not all components of the implanted FES systems are internal to the body where the stimulation electrodes are always implanted. These systems may include external sensors and/or a controller which communicate with the implant component. Common examples include cochlear implants, bladder management systems and neuroprosthesis for grasping. Percutaneous FES systems are those which have electrodes implanted in the body while the rest of the system is external to the body, i.e., the stimulation electrodes are inserted into the muscles and the leads that connect the electrodes to the stimulator penetrate the patient's

skin [13]. These systems have been used as orthotic devices and recently have been used to deliver FES therapy for walking [37].

## **2.5 Past and Current FES Systems**

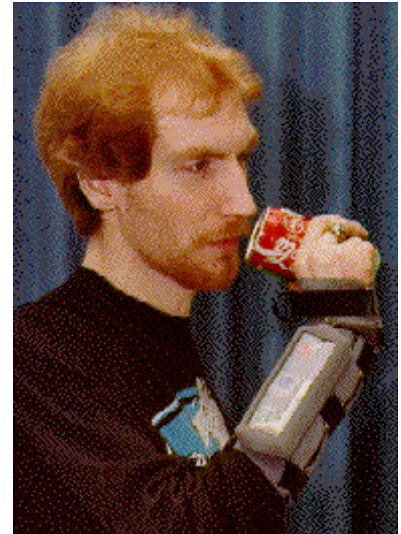
Applicable examples of past and current commercially available electric stimulators used for various FES applications include: Bionic Glove (Neuromotion Inc.), NESS H200 (BIONESS Inc.), ODFS<sup>®</sup> Pace (Odstock Medical Ltd), NESS L300 (BIONESS Inc.), WalkAide (Innovative Neurotronics), Parastep (Sigmedics Inc.), 300PV (EMPi), MOTIONSTIM 8 (Medel), Compex2 (Compex SA), Compex3 (CefarCompex), Freehand (NeuroControl), and the NeuRX DPS<sup>™</sup> RA/4 (Synapse Biomedical Inc.). These stimulation systems are divided into surface and implantable/percutaneous systems.

### **2.5.1 Surface FES Systems**

In surface FES systems, the entire system, including stimulating electrodes are external to the body. Surface stimulation is delivered using self-adhesive or non-adhesive electrodes, which come in a variety of shapes and sizes. These systems are non-invasive, electrodes are easy to apply, are generally less expensive and safer. However, since the stimulus signal must travel through skin, considerably higher-intensity signals are required due to the higher impedance of skin and dispersion of the signal when compared to subcutaneous or implanted stimulation electrodes. Typical current amplitude ranges are from 10 to 150 mA in surface stimulation [2]. Another limitation of surface stimulation is the targeting of deeper nerves and muscles such as the hip flexors. The following subsections provide further information on past and current commercially available surface FES systems.

### 2.5.1.1 Bionic Glove

The *Bionic Glove* developed by Neuromotion Inc is a four-channel neuroprostheses to assist with grasping [3]. Self-adhesive electrodes are placed over specific muscles. The glove is fitted over the forearm and wrist allowing the internal panels to make electrical contact with the electrodes. The system activates muscles to produce pinch-grip or hand opening functions among subjects with C5-C7 SCI and a small percentage of hemiplegic individuals who have some active wrist movement. Figure 2-10 shows the Bionic Glove in use during a pilot study conducted with C5-C7 SCI individuals [24, 38].



**Figure 2-10: Bionic Glove demonstration**

### 2.5.1.2 NESS H200

The *BIONESS NESS H200* is a non-invasive hand rehabilitation system intended as a neuroprosthesis for grasping in patients with neurological damage secondary to stroke, traumatic brain injury, and SCI [4]. The system consists of a specifically designed wrist hand orthoses with five surface electrodes for finger and thumb extensors and flexors, which are able to generate lateral and palmar grasp and release with stimulation [17]. A cable connects the orthosis to a portable control unit as shown in Figure 2-11.



**Figure 2-11: NESS H200 grasping neuroprostheses**

The control unit contains pre-programmed opening/closing stimulation patterns that the user can activate with push button controls. The patient is also able to increase or decrease grasping force using push buttons.

### 2.5.1.3 ODFS<sup>®</sup> Pace

The Odstock dropped foot stimulator (ODFS) by Odstock Medical Ltd is a small belt-worn single channel stimulator system shown in Figure 2-12 [5]. The stimulator is used as a neuroprostheses to correct the drop foot problem. The system is controlled with a foot-switch placed beneath the heel of a shoe. The switch provides the input for a timed electrical stimulation of the peroneal nerve.



**Figure 2-12: ODFS drop foot stimulator system**

### 2.5.1.4 NESS L300

The *BIONESS NESS L300* is a non-invasive drop foot stimulator [6]. It consists of a single electrode in a cuff designed to be donned by the patient. The device uses a set of sensors to detect stride position and state of gate cycle. This information is used by the control system (microcontroller) to calculate the appropriate stimulation trigger, level and PW (duty cycle). The entire system is

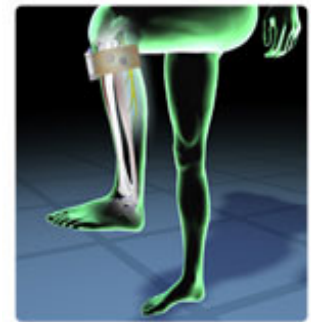


**Figure 2-13: NESS L300 drop foot neuroprostheses**

housed in a single cuff in Figure 2-13, which is strapped over the leg just under the knee.

### 2.5.1.5 WalkAide

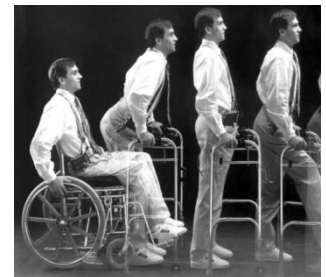
The *WalkAide* developed by Innovative Neurotronics is an external neuromuscular FES system illustrated in Figure 2-14, which is also used to correct the drop foot problem [7]. The WalkAide system is a battery operated single channel electrical stimulator, it utilizes a tilt sensor to control stimulation during normal gait. A clinician calibrates the system for a specific individual in order to make adjustments to the stimulation sequence based on acquired sensor data. This eliminates the need for a heel sensor during regular use.



**Figure 2-14:**  
**WalkAide on leg**  
**illustration**

### 2.5.1.6 Parastep<sup>TM</sup>

The *Parastep* from Sigmedics Inc. is a microcomputer controlled FES system that enables independent, unbraced ambulation (e.g., standing and walking) for people with a SCI [8]. The Parastep is a surface FES system with six stimulation channels, a microcontroller unit, a battery activated power pack with charger, a Paratester<sup>TM</sup> (testing unit for system operation), surface applied skin electrodes, power and electrode cables, and a stability walker with finger activated control switches. The surface electrodes are placed on the quadriceps, the gluteal muscles and the peroneal nerve. The user initiates each step while walking by



**Figure 2-15:**  
**Parastep system**

using the controls mounted on the walker shown in Figure 2-15. The Parastep obtained FDA (USA Food and Drug Administration) approval in 1994.

### 2.5.1.7 300PV

The *300PV* by EMPI, shown in Figure 2-16, is a two-channel, battery operated, portable neuromuscular stimulation device for multi-function electrotherapy [9]. The 300PV generates low level electrical stimulation to produce muscle contraction through surface stimulation electrodes. Adjustable parameters include type of waveform, pulse rate (frequency) and duty cycle (PW). Although this system represents a generic neuromuscular stimulator it does not have the capability of being applied as a custom neuroprosthesis due to the lack of stimulation programming flexibility and restriction to two stimulation channels.



**Figure 2-16: EMPI 300PV Stimulator**

### 2.5.1.8 MOTIONSTIM 8

The *MOTIONSTIM 8* by Medel is a surface stimulation FES system [10]. It is intended to operate as a generic FES system, being able to create stimulation programs with the appropriate software. The stimulator, shown in Figure 2-17, has a keypad with a display for choosing a stimulation program. Although



**Figure 2-17: MOTIONSTIM 8 stimulator system**

the stimulator can be programmed, it requires extensive engineering support in developing a custom stimulation program [39]. Also, the product is intended to be portable, but with its weight, dimensions, connector and cabling make it impractical as a portable system.

### 2.5.1.9 Compex2

The *Compex2* from Compex SA, shown in Figure 2-18, is a programmable FES system for surface stimulation using self adhesive electrodes placed on the patient's skin. The system has four stimulation channels and two analog/digital input channels. The Compex-2 is primarily developed for muscle strengthening and electromyography (EMG) based bio-feedback therapies.



**Figure 2-18: Compex-2 programmable FES system**

### 2.5.1.10 Compex3

The *Compex3* from CefarCompex, shown in Figure 2-19, is the next generation professional stimulator from Compex [11]. It features four stimulation channels with capability of multiplexing with sensors for capturing muscle activity. The system features a docking station which is used to charge or program the stimulator via a universal serial bus (USB) from a host PC.



**Figure 2-19: Compex-3 programmable FES system**

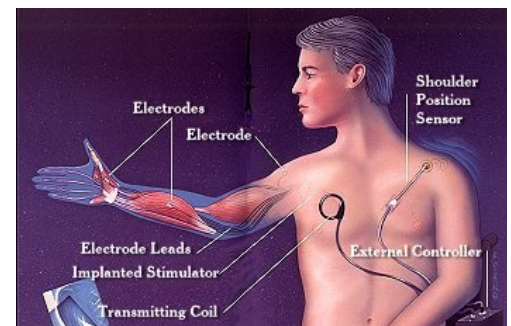


## 2.5.2 Implantable/Percutaneous FES Systems

With implantable and percutaneous FES systems, the stimulation electrodes are implanted directly on selected nerves and muscle tissues. Fully implantable FES systems also usually have an implantable controller circuitry with battery and have ability to communicate wirelessly to external systems for programming, control and or data acquisition. Percutaneous FES systems consist of electrodes with thin wires which are inserted through the skin directly into the muscular tissue. Compared to the fully implantable systems, these are used for a limited period of time (days or weeks). The advantages of implantable FES systems are higher stimulation selectivity and a considerably lower required electrical charge for muscle activation (usually under 25 mA). Drawbacks of such systems are that implants require a lengthy, invasive surgical procedure for installation, with greater risk of complications and infection. Also percutaneous systems can only be used temporarily due to higher risk of infection at penetration site. The following subsections provide an overview of past and current implantable FES systems.

### 2.5.2.1 Freehand

The *Freehand System* by NeuroControl was developed in Cleveland, Ohio and was approved for clinical use by the FDA in 1997. This system was commercially available until 2001 when the manufacture stopped manufacturing the systems [40]. The Freehand system featured an eight-channel stimulator implanted in the anterior chest wall, which was connected to the eight epimysial electrodes, as shown in Figure 2-20 [12]. The external components



**Figure 2-20: Freehand system component overview**



consisted of a sensor for detecting shoulder positions and a controller, which communicated to the implant through a coil taped to the skin over the implant. The movement of the contralateral shoulder proportionally controlled the degree of hand opening and closing [41]. The Freehand system had been implanted in over 250 individuals with C5-C6 tetraplegia, 51 of who participated in a multicentre clinical trial with positive results [42].

### 2.5.2.2 NeuRX DPS™ RA/4

The NeuRx Diaphragm Pacing System (DPS) by Synapse Biomedical Inc. consists of 4 percutaneous stimulation electrodes implanted into the diaphragm muscle and a fifth electrode placed under the skin which are all connected to the stimulator [13]. The stimulator provides the timing and control of stimulus to regulate movement of the diaphragm muscle to allow the patient to inhale. The stimulation parameters are programmed with the NeuRx Clinical Station™ that is connected and programmed by a clinician.



**Figure 2-21: NeuRX DPS™ RA/4 diaphragm pacing system**

### **2.5.3 Summary**

In summary, a considerable number of FES systems have been developed in the past, most of which work desirably in their target application. These systems were designed for a specific neuroprosthesis, which makes them impractical or normally impossible to adjust for other neuroprosthesis applications without major redesign of both hardware and software. Those stimulators that were developed as generic FES systems intended for use in various applications were limited in their use primarily due to restrictions in programming, stimulation capabilities, limited number of stimulation channels and user interface. Table 2-2 below provides a summary on the physical specifications of the stimulator units described above, these specifications relate to the performance and potential capabilities of each system.

Stimulator	# chan	Amplitude (mA)	Pulse- width ( $\mu$ s)	Stim. Frequency (Hz)	Battery life (Depending on use)	Waveform type
Bionic Glove	4	25 – 35	50– 200	20 – 30	N/A	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Rectangular</li> <li>• Biphasic/asymmetric</li> </ul>
NESS H200	5	Up to 150	10– 500	18 or 36	~ 15 Hrs	<ul style="list-style-type: none"> <li>• Constant voltage</li> <li>• Sinusoidal (11Khz)</li> <li>• Biphasic/symmetric</li> </ul>
ODFS Pace	1	20 – 100	7–365	40	2–3 weeks	<ul style="list-style-type: none"> <li>• Constant voltage</li> <li>• Biphasic</li> <li>• Symmetric/asymmetric</li> </ul>
NESS L300	1	0 – 80	100, 200, 300	20 – 45	N/A	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Biphasic</li> <li>• Symmetric/asymmetric</li> </ul>
WalkAide	1	115 @ 500 $\Omega$ 78 @ 1 K $\Omega$	50– 250	16.7 – 33	Up to 30 days	<ul style="list-style-type: none"> <li>• Constant voltage</li> <li>• Biphasic/asymmetric</li> </ul>
Parastep	6	0–300	150	24–25	100–120 min	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Biphasic/symmetrical</li> </ul>
300PV	2	0 – 100	50– 400	35–100	N/A	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Twin peak, mono/biphasic</li> <li>• Symmetric/asymmetric</li> </ul>
Motionstim8	8	0 – 125	10– 500	1 – 99	up to 10 Hrs	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Rectangular</li> <li>• Biphasic/symmetric</li> </ul>
Compex2	4	0 – 125	75– 16,00 0	1 – 100	~ 8 Hrs +	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Rectangular</li> <li>• Biphasic/symmetric</li> </ul>
Compex3	4	0 – 120	30– 400	1 – 150	~ 20 Hrs	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Rectangular</li> <li>• Biphasic/symmetric</li> </ul>
Freehand	8	2.5-20	0-200	16	N/A	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Rectangular</li> <li>• Biphasic/asymmetric</li> </ul>
NeuRX DPS RA/4	5	5 – 25	50– 200	5 – 20	500 Hrs	<ul style="list-style-type: none"> <li>• Constant current</li> <li>• Biphasic</li> </ul>

Table 2-2: Stimulator systems technical specifications

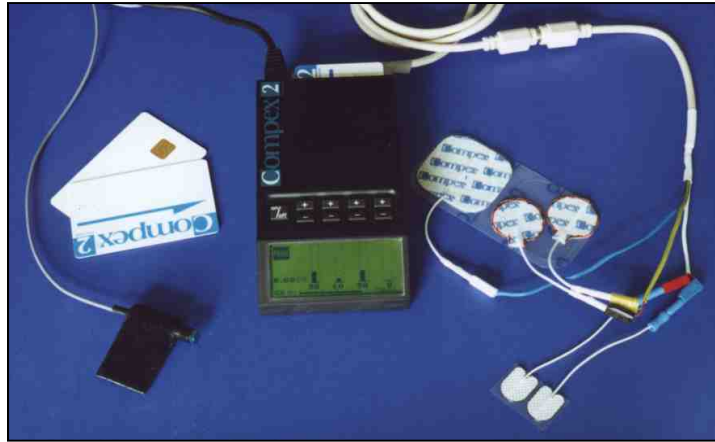
#

## **3 Proposed System**

The proposed FES stimulator represents the evolution of the Compex Motion stimulator system, which characterizes a proof of principle system created to demonstrate the feasibility of developing a fully multipurpose, programmable FES system. Over the past ten years the Compex Motion stimulator system was successfully used to test various design principles and ideas pertaining to FES orthotic and therapeutic applications. This acquired knowledge and expertise is now being used in part with the research and development of the next generation FES system.

### **3.1 Compex Motion Stimulator System**

The Compex Motion stimulator, shown in Figure 3-1 below, was created by Drs. Keller and Popovic in 2001 [36]. This system was developed in collaboration with company Compex SA, Switzerland and uses the same hardware platform as the Compex2 stimulator[36]. The Compex Motion was successfully applied as a neuroprosthesis for walking, reaching, standing and grasping in more than 100 stroke and SCI patients. This device was also used in animal experiments as a percutaneous FES system [43]. The stimulator was not commercially available because Compex SA decided not to expand into this area of the market.



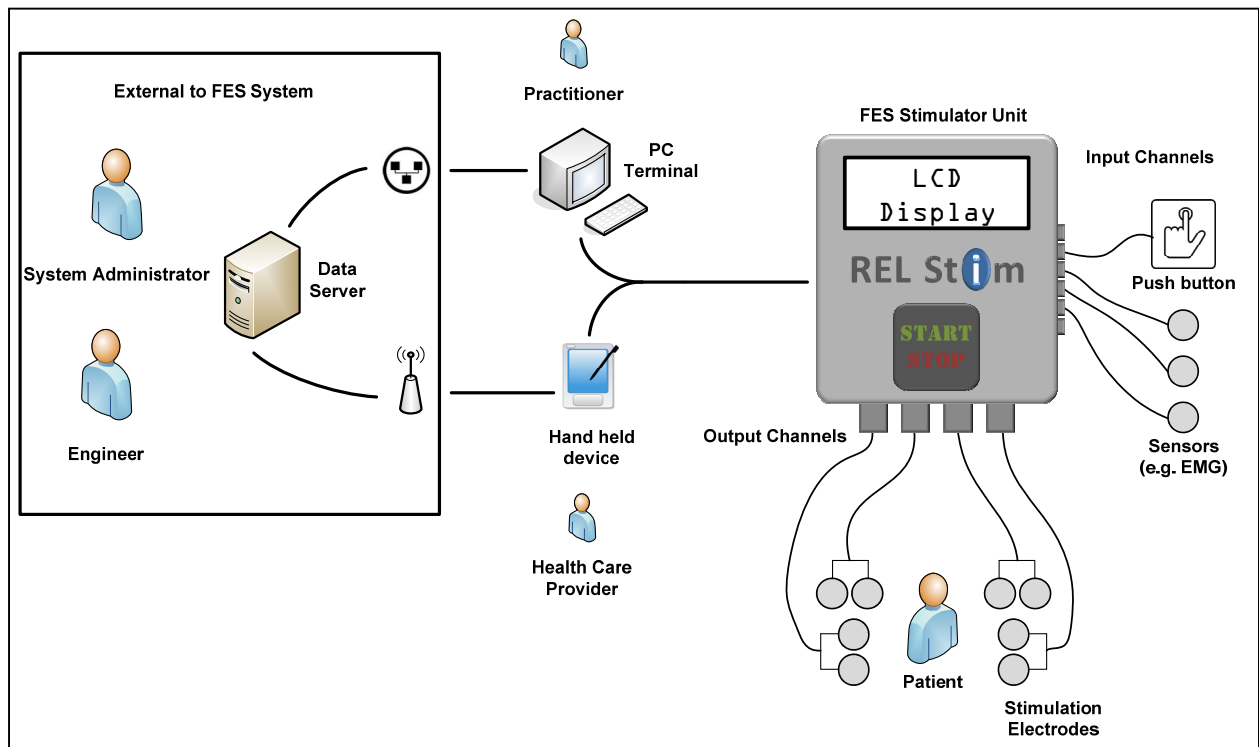
**Figure 3-1: Compex Motion Stimulator System**

The Compex Motion stimulator was designed as a hardware platform for the development of various neuroprostheses, which primarily deliver the stimulus using surface stimulation electrodes. The main characteristic of this system were that it provided a simple but powerful programming interface, which allowed practitioners to develop custom stimulation protocols and sequences. The system was also designed to allow the use of the same device with a number of different patients with unique stimulation protocols. In summary, the core features provided by Compex Motion included portability (powered by a rechargeable battery), four stimulation channels with expandability in sets of four (8, 12, 16, etc.), pulse amplitude, PW and frequency were independently controlled in real-time, stimulation channels were galvanically separated, and the stimulator had the capacity to be interfaced to external sensors, sensory systems or laboratory equipment for triggering and control of stimulation.

### **3.2 Concept System**

The proposed FES system shown in Figure 3-2 below is composed of the FES unit, which has four stimulation channels, expandable in multiples of four (4, 8, 12, 16, 20, etc.) and is able to record at least up to four input analog and four digital channels. The stimulator unit will

communicate to either a PC or hand-held device, both of which could be used to create custom-made stimulation programs for the stimulator through a simple to use but powerful graphical user interface (GUI) and provide direct stimulation control. These devices (PC/hand-held device) will also communicate to a central data server which maintains secure access to patient data and a stimulation program repository.

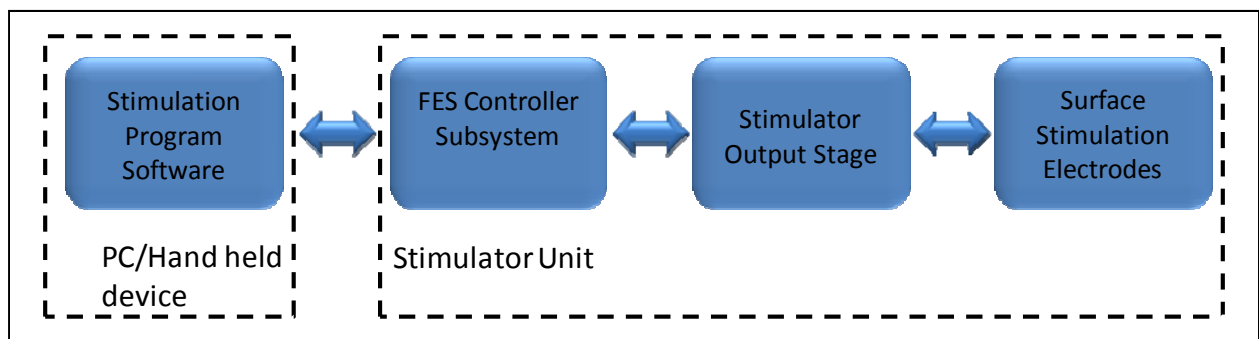


**Figure 3-2: High level diagram of the new generation FES system**

The proposed stimulator consists of two distinct subsystems. One subsystem represents the controller component of the stimulator and the second represents the output stage component of the stimulator. This thesis will deal with the research and development of the controller subsystem of the new FES system. This subsystem will: a) in real-time control the sequence and timing of the stimulation for each channel of the output stage, b) manage the various analog and/or digital input signals used to trigger and control the stimulator, c) acquire, process and

store recorded data (e.g., sensor data and stimulator status information), and d) communicate with a PC and/or hand-held device to allow the transfer of stimulation protocols and user control to start, stop, pause, resume, monitor and alter the stimulation.

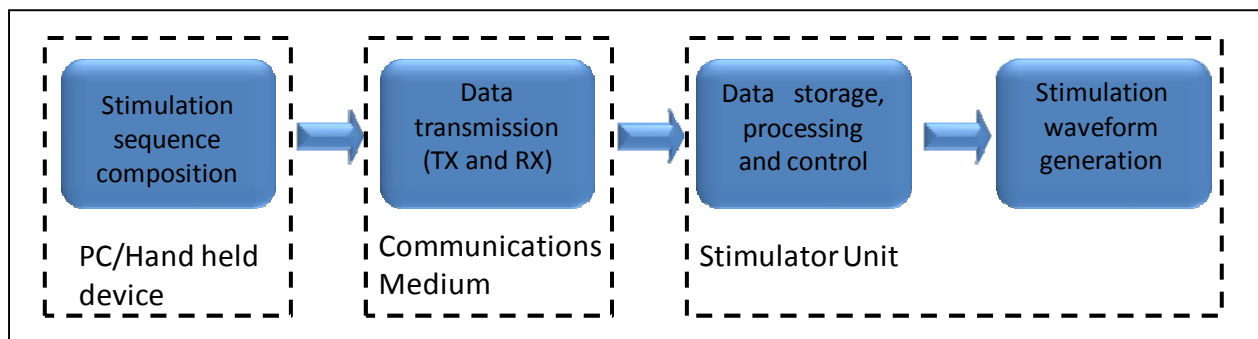
As can be seen in Figure 3-3 below, the overall FES system concept is broken down into four main components. The first is stimulation program and control software which would be used by a Health Care Provider (HCP), practitioner or researcher to create stimulation program files for various custom made neuroprostheses and to control the stimulator. Next, the FES controller subsystem component manages and controls the physical stimulator unit. The main functionality includes decoding and executing the stimulation program files with possible sensor input for triggering/control, communicating to a PC/hand-held device, interfacing to the output stage and the user. The stimulator's output stage component generates the physical FES stimulation waveforms and finally, the surface stimulation electrodes pass the generated stimulus to the targeted nerves and muscles in the patient.



**Figure 3-3: New generation FES system component diagram overview**

The high-level functional overview of this FES system is depicted in Figure 3-4 below. The first step involves the creation of an appropriate stimulation protocol for a given application. This would be done using the stimulation program software on the PC or hand-held device. Once the

stimulation program file has been created, it is transmitted to the stimulator unit (controller subsystem), which will then store the acquired program in memory. The user would then initiate the start of stimulation (unless the program is configured to start automatically the moment the stimulator is switched on) to begin the generation of the desired stimulation sequences. The stimulator would include further control from the PC/hand-held device including functionality to start, stop, pause, monitor stimulation parameters and change stimulation amplitude.



**Figure 3-4: New generation FES system functional diagram overview**

Various safety requirements have also been analyzed and included in the overall design and implementation. Some of these include having galvanically isolated stimulation channels in the output stage, ensure that during stimulation the stimulator is battery powered (i.e., the stimulation cannot be delivered when the stimulator is connected to a 110/240 VAC), ability to stop stimulation immediately (using the stimulator directly or the PC/hand-held device), and support of the maximum allowable PW and pulse amplitude for each channel separately.



### 3.3 Objectives

The work associated with this thesis includes the research and development of the control subsystem of the FES unit described above. Following the high-level system concept overview, the overall project objectives have been generated.

**Objective 1:** Determine the physical stimulation requirements the new stimulator must conform to, including supported stimulation frequency, PW and pulse amplitude ranges. Decisions will be based on capabilities of existing stimulator systems used for specific applications and research conducted in the effects of stimulation waveform characteristics. Since the new FES stimulator system is intended to be used to develop custom neuroprostheses, its stimulation capabilities must also conform to the requirements of the already existing FES applications.

**Objective 2:** Establish the programming and control capabilities the new stimulator must support. This will include type of stimulation pulses, required temporal resolution of stimulation protocols, and input sensor support. The stimulator must encompass the capabilities of being applied to simple applications such as wound healing and muscle strengthening, to complex applications such as synchronized activation of muscles that can generate motor control tasks such as grasping and walking.

**Objective 3:** Decide on the most appropriate hardware platform for the controller subsystem implementation. Decisions will be based on choosing a platform which has the performance and functional capacity based on the required functionality and that can be incorporated into a portable low-power device. The chosen hardware platform must also have the capabilities to conform to today's standards in functionality and performance.

**Objective 4:** Using the research results and decisions derived from Objectives 1 to 3, design and develop the controller subsystem for use in the new FES system. This will include requirement elicitation, system analysis, design, implementation and testing.

## **4 System Requirements and Analysis**

This chapter uses the information gathered in the background section of the thesis document to compile the overall system requirements for the controller subsystem. The system analysis is then derived from the requirements to provide the overall static structure and dynamic models of the system.

### **4.1 System Requirements**

This section provides the details pertaining to the system requirements, which have been established from the investigation of the current and past FES systems and their intended applications. FES systems have been designed and developed for specific applications such as grasping and drop foot neuroprostheses, and others designed for more generic FES applications. The controller being designed and developed as part of this project is intended for use in a universal FES system in order to provide the ability for one to adapt and apply the system to a wide variety of FES applications. For this reason, this novel FES system must encompass the physical characteristics of a large set of application-specific FES systems and support full programmability of stimulation protocols with the ability to interface to various input sensors.

#### **4.1.1 System Requirements and Analysis Tools**

The system modeling related tasks for this project have been analyzed and designed in part with some of the tools and techniques found in the Unified Modelling Language (UML) standard by Object Management Group (OMG) [44]. The UML standard had been chosen because it provided the means to a well-recognized and standardized process in system analysis and design,

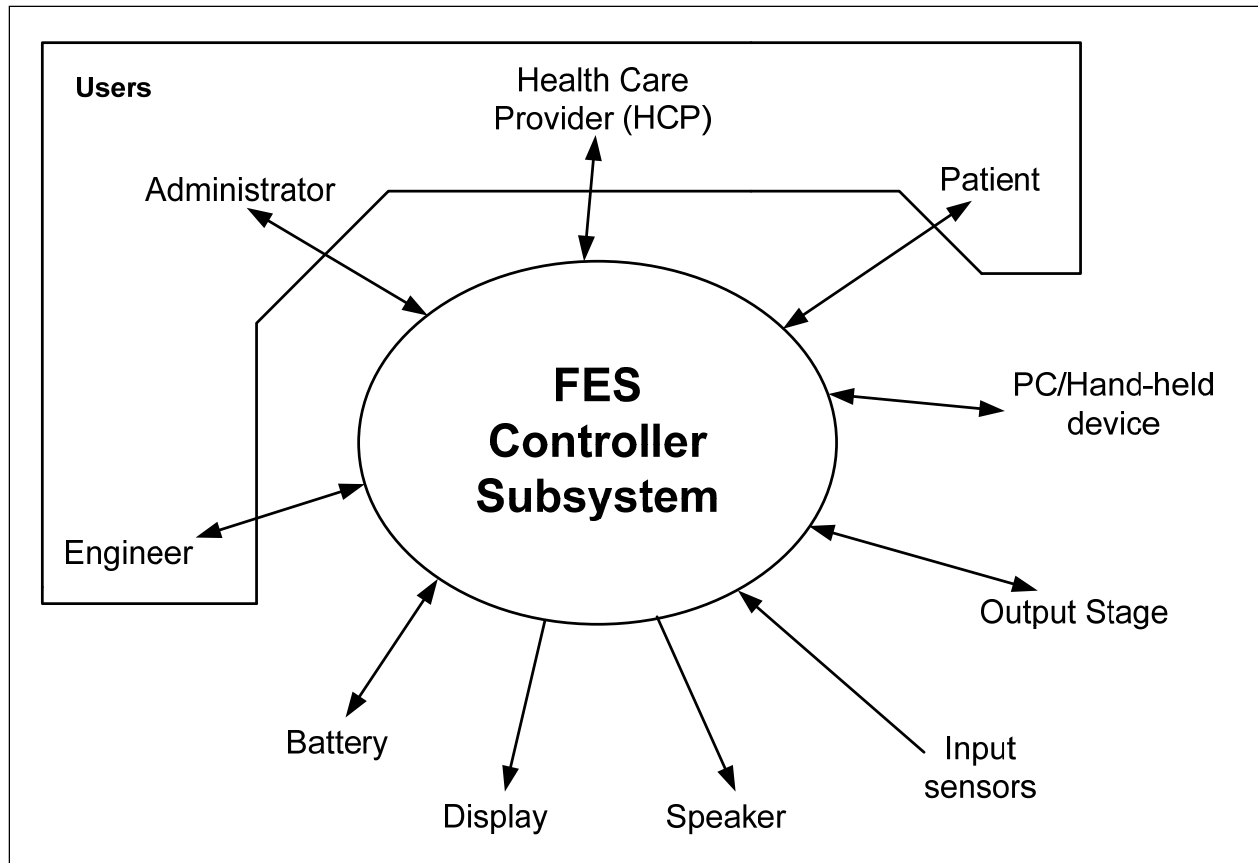
which facilitated structured development and function traceability for future maintenance. UML is a modeling language for specifying, visualizing, constructing and documenting the artefacts of the desired system in question [45]. It can be used as a guideline to realize a system similar to a “blueprint”. UML is a general-purpose, broadly applicable, tool-supported and industry standardized modeling language. The alternative to using UML as part of system analysis and design can be through using data flow diagrams, flow charts, Nassi–Shneiderman diagrams and Warnier-Orr diagrams, to name a few. As the uses of these tools have their own advantages for their specific application, UML provides not only the specific tools to realize different aspects of the system, but also the integration and links between the steps which provides traceability throughout the whole process.

Specifically for this project, UML tools and techniques were used for system requirement gathering, and system analysis. It has been used for the initial system modeling as the entire system design and development was started from ground zero. UML has provided an iterative approach with multiple visual representations of the desired system which was inspected by other team members including Drs. Milos R. Popovic, and Armen Baronian for feedback and opinions based on their expertise. The final requirement gatherings and system analysis is presented in this thesis, and has been used in the design and implementation in the controller subsystem.

### **4.1.2 System Context Analysis**

The first step in the requirement elicitation process was to identify all possible “actors” of the controller subsystem (stimulator). Actors include users (patients, engineers, HCPs and system administrators) and hardware components/devices that the controller subsystem has to interface

to (PC/hand-held device, output stage, sensors, speaker, display and battery). At the interface level, the boundaries of the controller subsystem are defined in Figure 4-1 below. This figure defines the interfaces between the system in question (controller subsystem) and all possible actors that might interact with it.



**Figure 4-1: Control subsystem context diagram**

Table 4-1 below defines the interfaces (actors) to the FES controller subsystem by describing their purpose and functionality at the system level with regards to the role each actor plays as part of the overall system.

Actor(s)	Description and Role
Users such as HCPs, patients, engineers, etc.	The users can be HCPs prescribing the system for patients, the patients who receive the FES intervention, and engineers who program, test, and maintain the system. These individuals could start and stop a stimulation sequence at any time directly from the stimulator unit and are able to observe basic system information.
PC/hand-held device	<p>A PC or a hand-held device, which is used by a user to communicates with the stimulator to:</p> <ul style="list-style-type: none"> <li>• Download stimulation protocol program to the stimulator</li> <li>• Upload stimulation protocol program from the stimulator</li> <li>• Display the stimulation protocol</li> <li>• Start, stop, pause, or resume stimulation</li> <li>• Adjust stimulation parameters in real-time</li> <li>• Display current stimulation parameters</li> <li>• Change or modify stimulation protocol</li> <li>• Gather and display sensor data and system statistics</li> </ul>
Output Stage	The output stage is the hardware that physically generates the waveforms of the stimulation pulses that are administered to the patient. The controller subsystem drives the output stage directly to produce overall desired stimulation. The output stage provides information on the status of the stimulation electrodes. In the event that an electrode fails or gets disconnected, the controller is informed and takes appropriate action such as pausing or stopping the stimulation protocol.
Input Sensors	Input sensors or sensory systems, which include both analog and digital sensors, can be used to interface the controller. Data obtained by the sensors is digitized and processed by the controller to be used as an input signal for the triggering and controlling a stimulation program. For example, a foot-switch can be used to trigger the stimulation sequence of a neuroprosthesis for walking. These sensors can be push buttons, pressure sensors, inclinometers, EMG, etc.
Display	The display will encompass visual notifications for the user about the stimulator's status. This will enable the user to quickly verify the current status of the stimulator including any error events that may occur while the system is running. This interface can also be used to display prompts for the user during a stimulation session.
Speaker	The speaker serves the purpose of providing audible sound(s) for the patient. This device can be incorporated within a stimulation protocol to, for example, prompt the patient about the imminent onset of the stimulation.
Battery	The interface to the battery provides controlled charging and battery level information back to the controller and user.

**Table 4-1: List of interface entities (users, devices, and other subsystems)**

### 4.1.3 List of Scenarios

The list of scenarios describes typical functionalities supported by the control subsystem. They specifically depict interactions between the control subsystem and its users, devices or other systems it interfaces to. A summary of typical scenarios supported by the system are detailed in Table 4-2 below, for a complete list of scenarios refer to Appendix 1: Requirements Models.

Scenario	Description
Download stimulation protocol	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to the controller for transmission of stimulation protocol to stimulator.</li> <li>Controller receives and stores stimulation protocol.</li> </ul>
Start stimulation	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to the controller with a request to start stimulation, or user instantiates start directly.</li> <li>Controller initializes the stimulator and starts the stimulation (assuming valid stimulation protocol).</li> </ul>
Adjust stimulation parameters	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller with request to change a stimulation parameter.</li> <li>Controller makes the appropriate parameter change within the ongoing (i.e., presently administered) stimulation protocol.</li> </ul>
Process stimulation protocol	<ul style="list-style-type: none"> <li>With timed execution, the controller decodes and processes the next stimulation parameters from loaded stimulation protocol.</li> <li>The final parameters for each stimulation channel are set for the output stage for each time instance within the stimulation.</li> </ul>

**Table 4-2: Sample list of scenarios**

### 4.1.4 Functional and Non-functional Requirements

Functional requirements describe the required characteristics and support of the controller subsystem and its environment. These specifications can be divided into two main groups; functional requirements, which detail what the system has to support, and non-functional requirements, which relate to usability, performance and reliability of the system.

The requirements compiled for this system are gathered from the research conducted in the field of FES systems and their applications. This overview is provided in the background section where stimulation characteristics with programmability configurations have been reviewed and various stimulators and stimulation systems have been analyzed with respect to their applications and capabilities. Since the controller for this stimulation system is intended to be applied to develop various custom neuroprostheses, its capabilities and performance must encompass that of most if not all of the application specific stimulation systems discussed in Section 2.5. The key difference in this system will be the ability to program and customize the system to adapt to various applications discussed above, where other stimulation systems are “application dedicated” and do not allow one to customize them for other applications or to develop fully customized stimulation protocols.

Table 4-3 below provides the summary of functional requirements and Table 4-4 contains the non-functional requirements of the controller subsystem. The requirements have been chosen to enable the necessary flexibility for the new generation stimulator system based on the research carried out as part of this thesis project. The stimulation characteristics requirements found in Table 4-4 are the desirable capabilities of the new integrated stimulator. In order for the stimulator to be developed as a whole, both the output stage subsystem and the controller subsystem must be able to meet these requirements. On the controller subsystem side, the necessary timed decoding and executions must be performed in order to support these stimulation characteristics as well as to provide some overhead extent for future improvements and broadened capabilities.



Functional Requirements	
Stimulation characteristics	User interface
<ul style="list-style-type: none"> <li>• Pulse amplitude: <ul style="list-style-type: none"> <li>○ 0-125 mA (1 mA step)</li> </ul> </li> <li>• PW <ul style="list-style-type: none"> <li>○ 0-16,000 <math>\mu</math>s (500 ns step)</li> </ul> </li> <li>• Pulse frequency <ul style="list-style-type: none"> <li>○ 1-200 Hz (1 Hz step)</li> </ul> </li> <li>• Number of channels <ul style="list-style-type: none"> <li>○ 4 per stimulator (option to append additional stimulation channels in multiples of 4, i.e., 8, 12, 16...)</li> </ul> </li> <li>• Waveform type <ul style="list-style-type: none"> <li>○ Rectangular (constant current)</li> <li>○ Monophasic/biphasic</li> <li>○ Symmetric/asymmetric</li> <li>○ Altering/non-altering</li> <li>○ Ability to change pulse properties every 100ms (timed based control) and after every delivered pulse (pulse based control)</li> </ul> </li> </ul>	PC/hand-held device interface
	<ul style="list-style-type: none"> <li>• Ability to start and stop directly</li> <li>• Visual display for stimulator status</li> <li>• Start/pause/resume/stop stimulation</li> <li>• Download stimulation program</li> <li>• Upload stimulation program</li> <li>• Change stimulation parameters</li> <li>• Change stimulation program</li> <li>• Transmit requested data <ul style="list-style-type: none"> <li>○ Current output stage parameters</li> <li>○ Input sensor data</li> <li>○ Stimulator status</li> </ul> </li> </ul>
Input sensor interface	Output stage interface
<ul style="list-style-type: none"> <li>• Process data from digital sensors (push buttons, switches, etc.)</li> <li>• Process data from analog sensors (sliding penetrometers, EMG, etc.)</li> <li>• Use processed data to trigger and control the stimulation</li> <li>• Capture and record input data</li> </ul>	<ul style="list-style-type: none"> <li>• Real-time stimulation control</li> <li>• Receive data on stimulation electrodes status <ul style="list-style-type: none"> <li>○ For detecting electrode failure or disconnection</li> </ul> </li> <li>• Galvanically isolated channels</li> </ul>
Stimulation Programmability Support	
<ul style="list-style-type: none"> <li>• Constant stimulation waveform <ul style="list-style-type: none"> <li>○ Constant Amplitude and PW</li> <li>○ Configurable length of time (ON-time)</li> </ul> </li> <li>• Ramp up and ramp down <ul style="list-style-type: none"> <li>○ Gradually increase/decrease PW from present state to new value</li> <li>○ Configurable ramp time</li> </ul> </li> <li>• Instantaneous changes in stimulation parameters <ul style="list-style-type: none"> <li>○ Change PW, amplitude and frequency</li> <li>○ Programmable at any point in stimulation protocol</li> </ul> </li> <li>• Loops <ul style="list-style-type: none"> <li>○ Looping of specific stimulation tasks</li> <li>○ Configurable loop region and number of loops (including nested loops)</li> </ul> </li> <li>• User interactions <ul style="list-style-type: none"> <li>○ Trigger based on user input (analog or digital inputs)</li> <li>○ Configurable active trigger location and criteria</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• No stimulation <ul style="list-style-type: none"> <li>○ Zero PW</li> <li>○ Configurable length of time (OFF-time)</li> </ul> </li> <li>• Channel synchronization <ul style="list-style-type: none"> <li>○ Enable all channels to proceed from specified point at the same time</li> <li>○ Configurable synchronization point(s)</li> </ul> </li> <li>• Parameter randomization <ul style="list-style-type: none"> <li>○ Randomize PW, amplitude and frequency</li> <li>○ Randomization range can be set between 0-100%</li> <li>○ Configurable On and OFF points where randomization is enabled and disabled, respectively</li> </ul> </li> <li>• User alerts <ul style="list-style-type: none"> <li>○ Audible and visual alerts</li> <li>○ Configurable sound and textual alerts</li> </ul> </li> </ul>

<ul style="list-style-type: none"> <li>Configurable safety limits for channel-specific PW and amplitudes</li> </ul>	<ul style="list-style-type: none"> <li>No stimulation during battery charge</li> </ul>
---	--

**Table 4-3: Functional Requirements**

The non-functional requirements listed in Table 4-4 were primarily utilized during the evaluation of the appropriate hardware and electrical components of the system. These requirements provided the minimum physical and performance characteristics that the system had to meet when final integration and assembly took place.

Non-Functional Requirements	
Controller Hardware	User interface
<ul style="list-style-type: none"> <li>Operating temperature <ul style="list-style-type: none"> <li>At least 5 °C to 45 °C</li> </ul> </li> <li>Power Control <ul style="list-style-type: none"> <li>Options for idle, sleep or power down modes</li> </ul> </li> <li>Flash memory <ul style="list-style-type: none"> <li>At least 256 kB</li> </ul> </li> <li>Processing clock speed <ul style="list-style-type: none"> <li>At least 30 MHz</li> </ul> </li> <li>Analog to digital converter (ADC) <ul style="list-style-type: none"> <li>At least 4 inputs</li> <li>At least 8-bit resolution</li> <li>At least 8kHz sampling rate</li> </ul> </li> <li>Digital to analog converter (DAC) <ul style="list-style-type: none"> <li>At least 1 output channel</li> <li>Minimum 8-bit resolution</li> </ul> </li> <li>Portability (overall packaging requirements) <ul style="list-style-type: none"> <li>Run on battery for at least 12 uninterrupted hours</li> <li>Overall weight less than 500g with batteries</li> <li>At most 148mm length, 80mm width, and 30mm height</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Start and stop stimulation <ul style="list-style-type: none"> <li>One user input button</li> <li>Red in color</li> <li>At least 2 cm in diameter</li> </ul> </li> <li>Stimulator status display <ul style="list-style-type: none"> <li>At least two line with 15 character per line</li> </ul> </li> </ul>
	PC/hand-held device interface
	<ul style="list-style-type: none"> <li>Use standard interface connection for maximum compatibility</li> <li>Provide sequential request handling scheme interface</li> <li>Support of single PC/hand-held device interface at a time</li> </ul>
	Output Stage Interface
	<ul style="list-style-type: none"> <li>Modular design with variable stimulation channels as required</li> </ul>

**Table 4-4: Non-Functional Requirements**

### 4.1.5 Use Cases

Each use case is an abstraction from the scenarios described above, they specify all possible situations and actions required by the system for a given component or function. These models provided a detailed overview of what the system in question was expected to do in relation to the input from and interactions with the external actors. This information was derived through literature review of the contemporary FES systems and their applications (see Section 2.3 for the characteristics of FES and Section 2.5 for a review of applicable FES systems) and is used for the system design, analysis and testing. Below, two use cases are provided as examples, the full set of test cases can be found in Appendix 1: Requirements Models in Table A1-4 to Table A1-14. The first example use case describes the scenario of starting a stimulation sequence and is detailed in Table 4-5 below. In this scenario in order to be able to start a stimulation sequence, a stimulation protocol had to be loaded and ready within the controller. At this point the stimulation can be started from the user by pressing the start button on the stimulator itself or by using the PC/hand-held device to initiate a start command.

The “Alternate flows” row in the use case description tables provided information on the conditions that might cause deviations from the ordinary execution path. For example, in the use case detailed in Table 4-5, step 2 in the “flow of events” row, its execution path could be disrupted if a stimulation program was not loaded, or if a stimulation sequence was already being executed by the controller. These two conditions could alter the regular execution path, and therefore were described in the “Alternate Flows” section of the use case.

<b>Use case name</b>	Start Stimulation
<b>Primary actor(s)</b>	HCP/Patient/Administrator/Engineer – user or PC/hand-held device
<b>Secondary Actor(s)</b>	Output Stage
<b>Pre-condition</b>	1. The user or PC/hand-held device has initiated the start of the stimulation.
<b>Flow of events</b>	2. The controller initializes stimulation parameters based on protocol and configures the output stage 3. Stimulation status is updated and this information is display on the stimulator
<b>Post-condition</b>	4. Stimulation sequence execution has started.
<b>Alternate Flows</b>	2.1 Stimulation program file is not loaded in memory. 2.2 Controller updates status and displays error message for user. 2.3 Stimulation is not started. 2.1 Stimulation already running 2.2 If the command arrives from the stimulator – this indicates a stop 2.3 If the command arrives from the PC/hand-held device – the stimulation continues
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	The controller is in idle or ready mode.

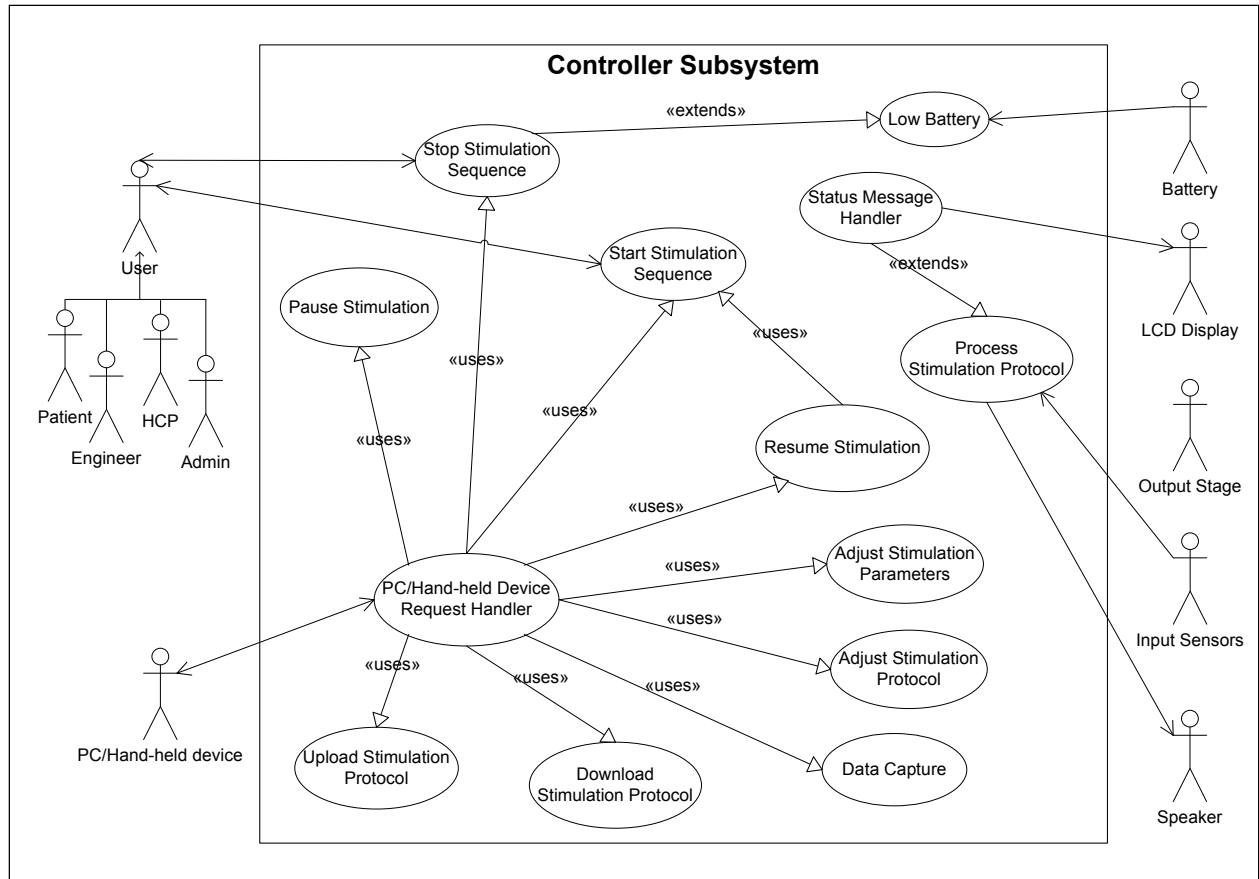
**Table 4-5: Use case start stimulation**

The second sample use case scenario is that of downloading a stimulation protocol to the stimulator, which is described in Table 4-6 below. This function would be performed from the PC/hand-held device, which establishes a connection request to the stimulator to download a new stimulation protocol. The protocol is then stored within the controller, and is ready for further processing and execution. At this point the stimulator is ready to start processing and executing the loaded stimulation protocol. The downloaded protocol can also have the option to be configured to start immediately after the download and downloaded protocol validation process. The protocol validation process tests for any possible transfer errors, which can appear in the stimulation protocol.

<b>Use case name</b>	Download stimulation protocol
<b>Primary actor(s)</b>	PC/hand-held device
<b>Secondary Actor(s)</b>	HCP/Patient/Administrator/Engineer – user
<b>Pre-condition</b>	1. The PC/hand-held device establishes a connection to download a stimulation protocol to the stimulator.
<b>Flow of events</b>	2. The controller acknowledges a valid connection and proceeds with the request. 3. The PC/hand-held device transmits entire stimulation protocol to controller. 4. The controller validates and stores the stimulation protocol.
<b>Post-condition</b>	5. The controller updates its status and displays an appropriate message.
<b>Alternate Flows</b>	2.1 Current stimulation running. 2.2 The controller rejects the request from the PC/hand-held device. 3.1 Error detected in received stimulation program file. 3.2 The controller updates its status and displays an appropriate error message for the user.
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	The appropriate connection is made to the PC/hand-held device.

**Table 4-6: Download stimulation protocol**

Once all use cases for the required functionality and components are defined, a use case diagram is constructed to provide a visual representation of all the use cases, and the relationships between use cases and the actors of the system. This model provides insight on the connections and interconnections of the system; this information is valuable for the analysis stage as dependencies can be identified and described. Figure 4-2 below depicts the use case diagram for the controller subsystem.



**Figure 4-2: Controller subsystem use case diagram**

## 4.2 System Analysis

The overall objective of the system analysis was to transform the specifications of the system in a form ready for design and implementation. The specifications of the system are detailed in the requirements elicitation procedure provided above which resulted in the use case model of the system. The analysis representation of the system resulted in the static and dynamic UML models which are described in the following subsections.

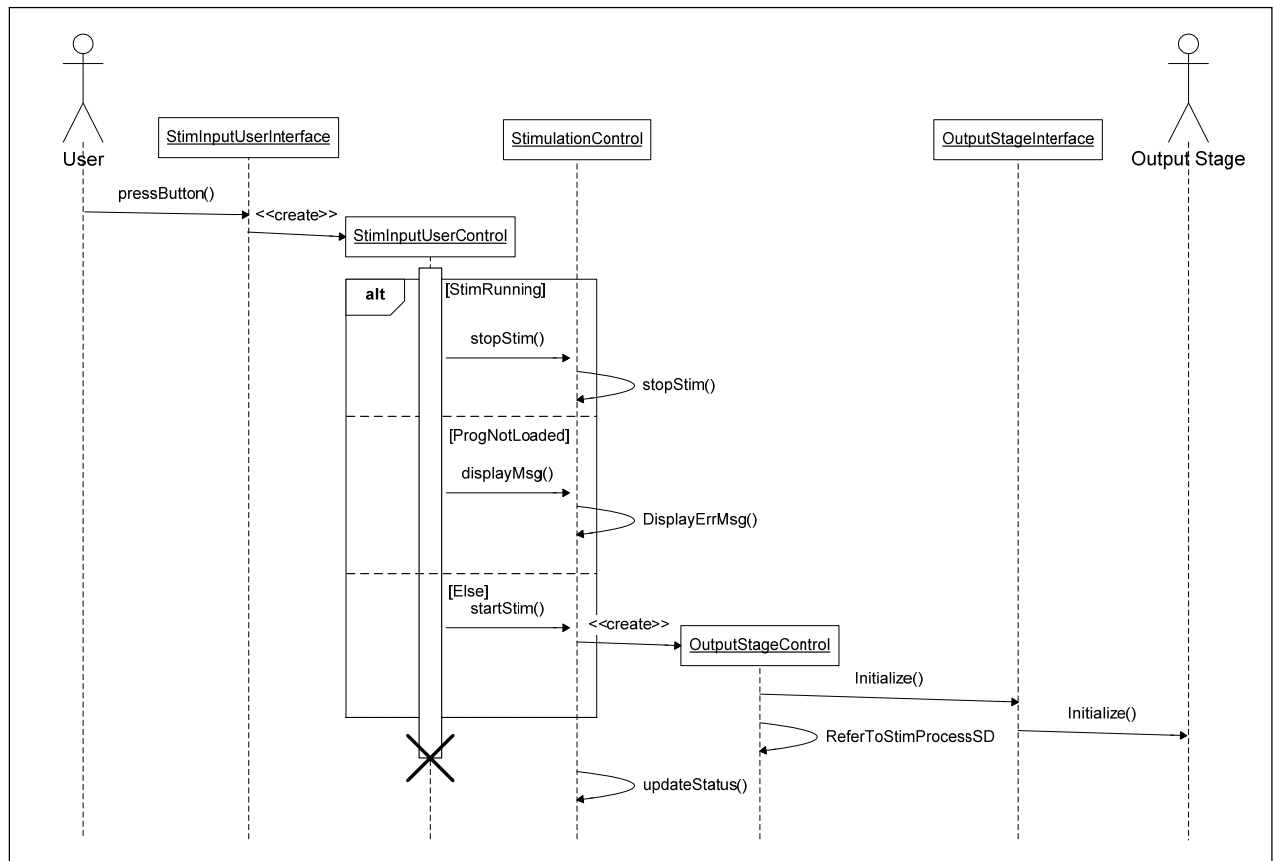
### **4.2.1 Static System Model**

The static system model described the structure information about the system. This model was created through the composition of a class diagram, which provided the possible static solution for the controller subsystem. The first step in the development of this model was to identify all objects which make up the controller subsystem. These objects were extracted from the use case model. Once the objects were identified, they were grouped into three categories; entity objects which represent persistent information tracked by the system, boundary objects which represent the interface between the system and specific actors, and control objects which represent the logic and interactions between objects. Then the relationships between the identified objects were identified, in terms of which objects would interact with each other, to exchange information or make use of specific functionality. Finally the attributes which denoted the properties of individual objects were identified. These attributes included variables associated with each object and methods or functions that each object was responsible for. The resulting full UML class diagram can be found in Appendix 2: Static System Model.

### **4.2.2 Dynamic System Models**

The dynamic system models provide details on the system's behaviour, this stems from the static model by analyzing the execution-time behaviour of the identified objects and interactions between these objects. The dynamic system models continue from the use case descriptions by showing the behaviour of each object pertaining to the specific use case. These models are provided through the use of sequence diagrams which show how the behaviour of a use case is distributed over its participating objects and actors. These sequence diagrams provide further details for the design and implementation stage of the project. This information is used to better

understand how the system's functionality will be realized. Figure 4-3 below displays the sequence diagram for starting a stimulation sequence by the user directly. In this use case, the user activates the button on the stimulator unit to start the stimulation. This action is received through the user interface boundary object which then instantiates the appropriate control object to perform the required action (in this case start of stimulation). The control object verifies the request by retrieving the current state of the stimulator and triggering the appropriate action. For example, if the stimulation is already running, a stop is issued. If the stimulation is stopped with a valid stimulation protocol loaded, the output stage control object is created, the output stage is initialized and the stimulation would be started.



**Figure 4-3: Start stimulation (user) sequence diagram**



## **5 Hardware and Software Platform Overview**

This chapter walks through the initial hardware and software design decisions used to develop the controller subsystem. The chapter is broken down into two main parts: 1) decisions on hardware and software platforms used for the system, and 2) the overall software structure of the controller subsystem. When deciding on the hardware platform for use in this system, applicable alternatives were compared and considered, and the final decision was based on the most suitable option. Similarly, once the decision on the hardware platform has been made, various options in software platforms and support tools were considered and an appropriate decision was made and is described. Finally the initial overall software structure of the controller subsystem is provided.

### **5.1 Hardware Platform**

From the requirements analysis, the chosen controller hardware platform must meet the specifications found in Table 5-1 below. Two different hardware platform categories have been considered when deciding the most suitable option, these included: 1) microcontrollers (such as ARM (Advanced RISC (Reduced Instruction Set Computer) Machine), Atmel AVR (Advanced Virtual RISC) and PIC (Peripheral Interface Controller) microcontrollers); and 2) Field Programmable Gate Arrays (FPGAs) (such as Altera and Xilinx). Other questions which had to be considered when deciding on a specific hardware platform and device included: a) how much will the end product cost? and b) what are the input, output and overall integration requirements of the system?

Controller Hardware
<ul style="list-style-type: none"> <li>• Operating temperature <ul style="list-style-type: none"> <li>○ At least 5 °C to 45 °C</li> </ul> </li> <li>• Power Control <ul style="list-style-type: none"> <li>○ Options for idle, sleep or power down modes</li> </ul> </li> <li>• Flash memory <ul style="list-style-type: none"> <li>○ At least 256 kB</li> </ul> </li> <li>• Processing clock speed <ul style="list-style-type: none"> <li>○ At least 30 MHz</li> </ul> </li> <li>• ADC <ul style="list-style-type: none"> <li>○ At least 4 inputs</li> <li>○ At least 8-bit resolution</li> <li>○ At least 8kHz sampling rate</li> </ul> </li> <li>• DAC <ul style="list-style-type: none"> <li>○ At least 1 output channel</li> <li>○ At least 8-bit resolution</li> </ul> </li> <li>• Portability (overall packaging requirements) <ul style="list-style-type: none"> <li>○ Run on battery for at least 12 uninterrupted hours</li> <li>○ Overall weight less than 500g with batteries</li> <li>○ At most 148mm length, 80mm width, and 30mm height</li> </ul> </li> </ul>

**Table 5-1: Controller hardware platform minimum requirements**

The decision process started by deciding on using either a microcontroller based or FPGA based architecture. The design and implementation stage differ greatly depending which of these two architectures is chosen. The structure of a microcontroller is comparable to a small computer placed in a single chip. This chip has all the necessary components found in a computer such as processor, memory, timers, and various I/O. They are used in almost all embedded devices which require to perform specific tasks and which connect to other devices or computers for communications. Overall microcontrollers are intended for use in low-power applications, making them well suited for devices with a battery source. In general microcontrollers are programmed using software. The summary of the advantages and disadvantages of using a microcontroller based platform are listed in Table 5-2 below.

<b>Microcontrollers</b>	
<b>Advantages</b>	<b>Disadvantages</b>
Power	Unused functionality
Programming (ASM, C, C++)	Initial design flexibility
Future design flexibility	Lower performance
Standardized device support	

**Table 5-2: Summary of advantages and disadvantages of microcontroller architectures**

FPGAs are integrated circuit designed devices that are configured for specific task(s) by the designer – hence the term “field-programmable”. These integrated circuits can contain millions of logic gates which can be “connected” or configured to a desired specification. The nature of FPGAs allow them to be very flexible since you start off with arrays of gates, these can then be used to perform any logic function that can fit within the provided number of gates. In contrast, a microcontroller already has its own circuitry, instruction sets and protocols that have to be used in the design. FPGAs are programmed in hardware using a hardware description language (HDL). Overall the flexibility and performance of FPGAs comes at a cost as they usually consume more power and are more expensive than a typical microcontroller (see Table 5-3 below for its advantages/disadvantages).

<b>FPGAs</b>	
<b>Advantages</b>	<b>Disadvantages</b>
Higher performance	Power
Initial design flexibility	Future design flexibility
Greater accuracy and timing	Higher cost
Design for req. functionality	Longer time to market

**Table 5-3: Summary of advantages and disadvantages of FPGA architectures**

Comparing the two platforms described above, specifically that of a microcontroller based system and an FPGA based system, the advantages and disadvantages when using one over the other have been summarized in Table 5-2 and Table 5-3 above. From this analysis, it can be summarized that microcontroller based architectures provide a more standardized design approach using protocols, methods and building blocks (e.g., analog to digital converters (ADC), digital to analog converters (DAC), timers, universal asynchronous receiver transmitters (UART), etc.) that have been industry tested and proven in functionality and reliability. The very nature of microcontrollers allows the possibility for future design and development for additional support and features without necessarily re-designing the entire system. In addition, most microcontrollers provide integrated power saving modes that allow the system to be placed in idle or sleep thus lowering power consumption compared to constant power utilization under a full operational mode.

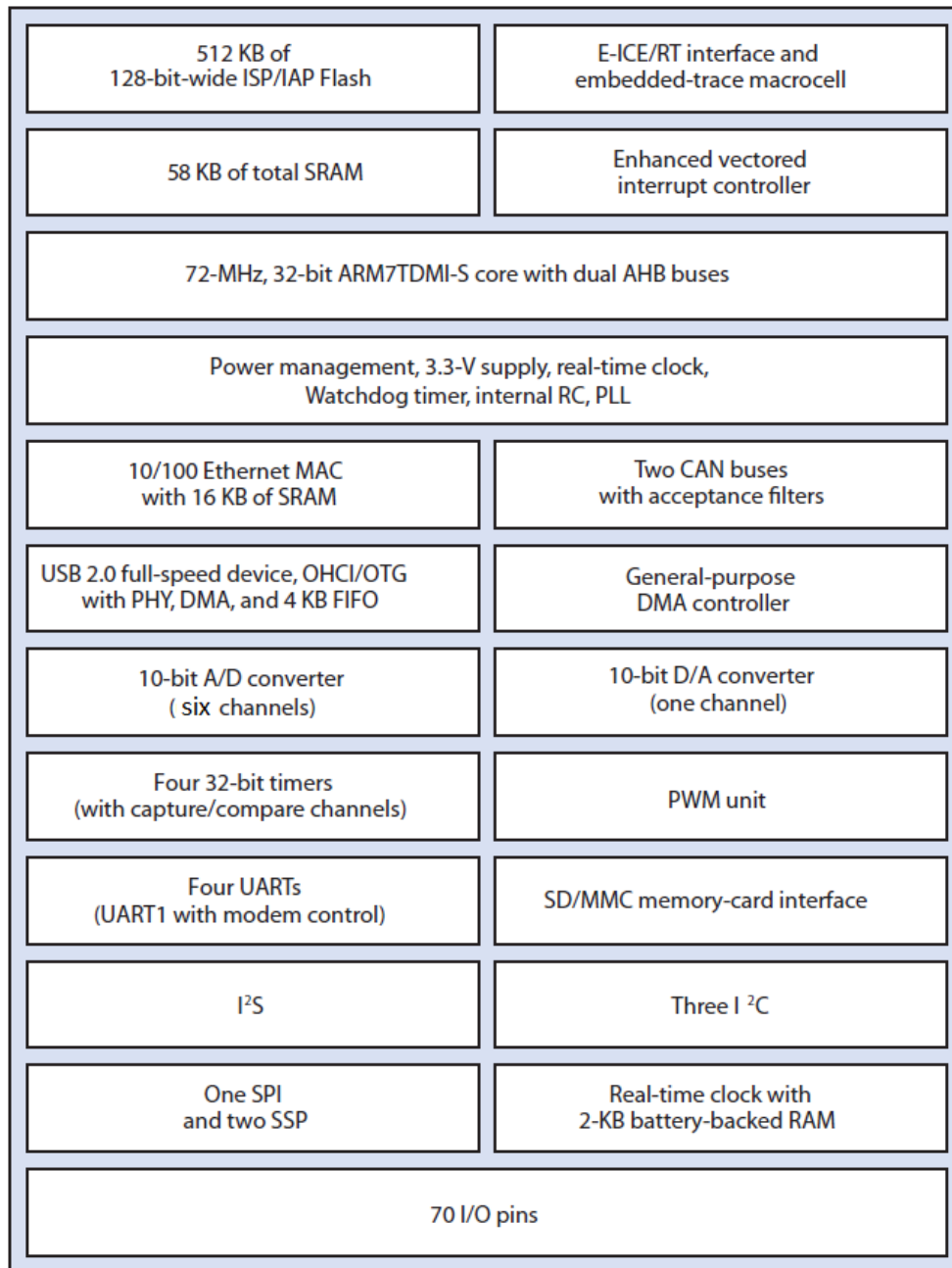
On the other hand, an FPGA based system in general allows for higher performance which is best suited for complex algorithms or digital signal processing (DSP) calculations where results must appear within short windows of time. Furthermore, designs using this type of platform allow for maximum initial design flexibility, allowing one to design and implement only the required and exact functionality. When using a microcontroller, the most suited device for an application is chosen but in most cases there are always features and functionality that are included but never used or required. After considering the advantages and disadvantages of both hardware platforms for use in the controller subsystem design and implementation, the microcontroller architecture was selected as the best choice. The main deciding factors favouring a microcontroller architecture were: a) lower power consumption with greater power control/usage options making it ideal for use in a portable system powered by batteries, b) ability

to be more flexible for future design changes and feature support, c) inclusion of industry standard communication blocks, and d) its lower cost with shorter time to market advantage.

When deciding on the appropriate microcontroller device, the minimum requirements listed in Table 5-1 have to be met with a platform which ideally also allows the possibility to evolve and build on the initial system design in the future. Various common microcontroller architectures which are frequently found in the portable device industry included the ARM, Atmel AVR and PIC microcontrollers. The ARM architecture is the chosen platform as it is intended for higher-performance systems where multiple functions and roles are supported by a single device. In contrast AVR and PIC architectures are generally intended for use in systems performing fewer parallel processes at slower rates when compared to ARM.

The ARM processor family chosen for the controller subsystem is the ARM7TDMI-S processor, which is a 32-bit embedded RISC processor. This ARM family allows design and development of embedded devices requiring small size, low power and high performance. The ARM7TDMI platform has been shown to be an ideal architecture for use in a cochlear implant stimulation system [46], where small size, ultra low power requirements and enhanced performance were required. As the cochlear implants and our FES system have many commonalities in function and performance requirements, it was natural to select a microcontroller that has been already proven in similar applications. Within the ARM7TDMI-S processor family, the LPC2368 microcontroller developed by NXP (founded by Philips) was chosen. The LPC2368 microcontroller has features including: a) a processor capable of running up to 72 MHz; b) 512kB of on-chip flash memory; c) various reduced power mode options; d) a 10-bit ADC multiplexed among 6 pins at a sampling frequency of up to 4.5 MHz; e) a Secure Digital/Multi

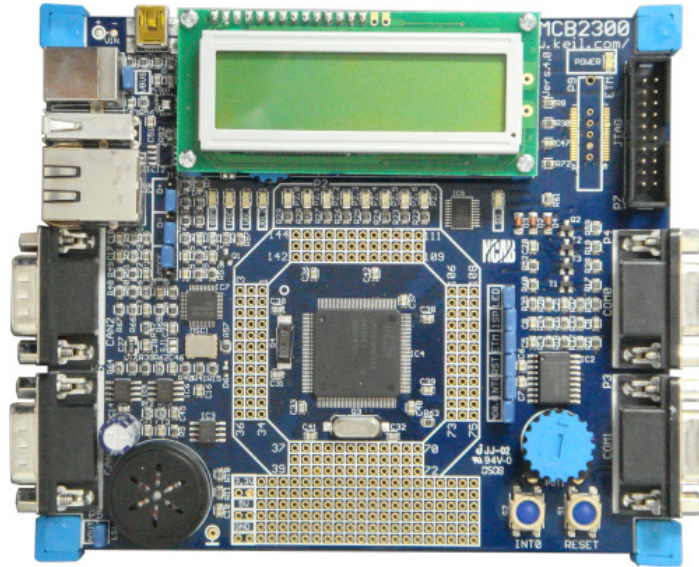
Media Card (SD/MMC) memory card interface and f) an operating temperature range of -40°C to 85°C. Developing the controller subsystem with this platform allows one the option to upgrade the platform in the future up to the ARM Cortex-M3 platform if greater performance and or additional features are required by the controller subsystem [47]. The block diagram for the LPC2368 microcontroller is shown in Figure 5-1 below. This figure contains the high-level specifications and functional features supported by this microcontroller.



**Figure 5-1: Block diagram of the LPC2368 microcontroller (adapted from [48])**

For the purpose of this project, the MCB2300 evaluation board (Keil™, an ARM® company, USA) populated with the LPC2368 microcontroller was used for the controller subsystem development, and can be seen in Figure 5-2 below. The use of this evaluation board lead to a quick start with the development as it allowed us to program and test the new microcontroller

architecture without physically having the stimulator's microcontroller driver board fully developed and finalized.



**Figure 5-2: MCB2300 evaluation board with LPC2368 microcontroller (adapted from [49])**

## 5.2 Real-Time Software Platform and Development Toolset

The development tool set chosen for working with the LPC2368 microcontroller is the ARM Microcontroller Development Kit (MDK-ARM, Keil™, an ARM® company, USA). This development kit was chosen because it provides a complete industry standard development and debugger package support for ARM microcontrollers, which was developed and supported by directly by ARM®. This development kit includes the ARM C/C++ compiler with support for ARM7, ARM9, Cortex-M0, Cortex-M1, and Cortex-M3 devices. Included in this kit is the µVision integrated development environment (IDE) with debugger, simulation environment and the Keil RTX deterministic Real-Time Operating System with hard-real time support.

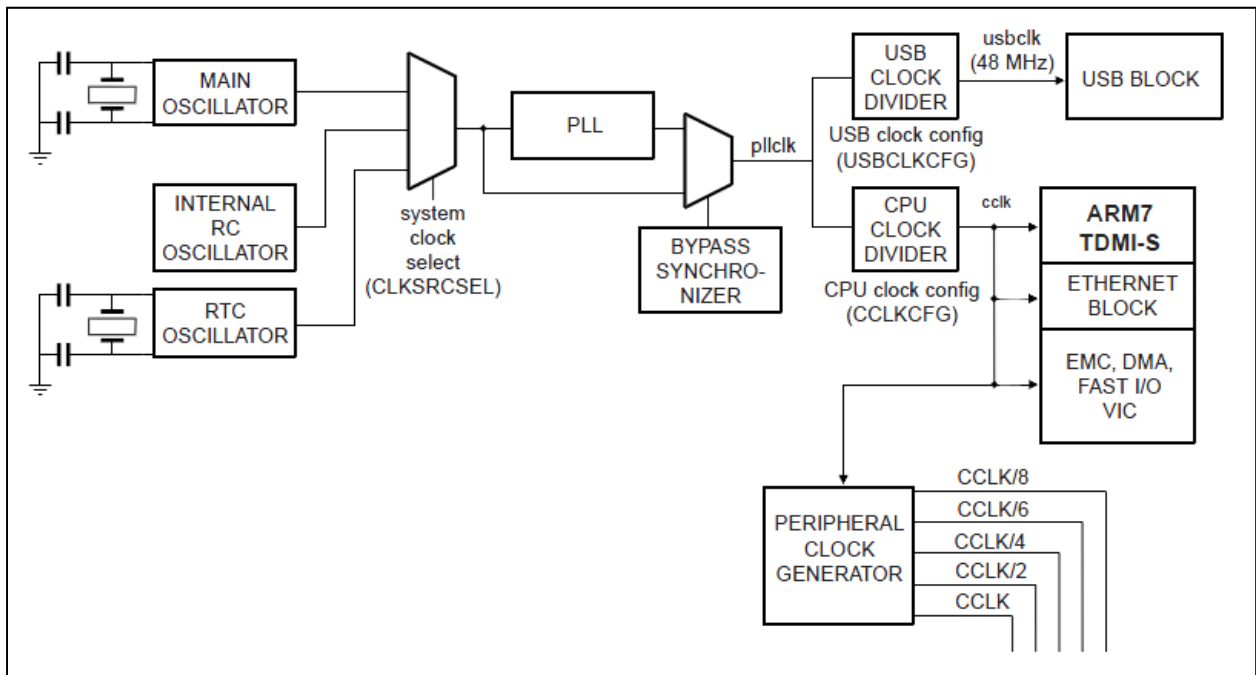


A real-time operating system (RTOS) was chosen for use in the development of the controller subsystem due to the overall system complexity and necessity to execute multiple processes and tasks with a reliable scheduler. The use of an RTOS allows support for scheduling, timing and maintenance of all processes which make up the system in development. Therefore, the use of a well-established RTOS was required. The Keil RTX RTOS was chosen as it was royalty-free with source code, designed and supported by ARM, provides real-time operation support with flexible scheduling options for processes and tasks, and has a small memory foot-print with a code size of less than 4.2 KB [50].

### **5.3 System Structure Overview**

The structure for the overall system platform extends from the system analysis discussed in Section 4.2. The real-time software structure was implemented with the Keil RTX real-time kernel and consisted of five core classes. These classes included: a) the main system class, entitled “projectMain”, which starts the execution of the controller subsystem by initializing all global variables and all core system components; b) the “pcControl” class implemented the interface which was used to communicate with a PC/hand-held device (all communication requests are handled by this class) (Chapter 6); c) the “displayControl” class that provided the interface to write various messages including status, error and stimulation specific messages with timing options for the user display on the stimulator (Section 7.1.2); d) the “stimulationProgram” class managed the stimulation protocol stored within the microcontroller, access to the stimulation protocol including writing and reading parts of a stimulation protocol was done through this class (Section 9.3); and e) the “outputStageControl” class dealt with the real-time processing, decoding and execution of a stimulation protocol to generate the desired stimulation sequences at the output stage (Section 9.3).

The central processing unit (CPU) clock on the LPC2368 microcontroller was configured to operate at 48 MHz (CCLK – CPU clock operating frequency) in order to provide the appropriate clock division results for the various components required in this design, such as the USB and UART blocks. The clock generation diagram for the LPC2300 series of microcontrollers is provided in Figure 5-3 below. The selected clock source (CLKSRC) was the main oscillator provided on the evaluation board, which operated at 12 MHz. This clock value was fed into the Phase Locked Loop (PLL) block which multiplied the input clock to a high frequency and then divided down to provide the actual clock used by the CPU and USB block. The resulting PLLCLK value was 240 MHz, which was generated by multiplying the input clock (12 MHz) by 20. This value was then divided by five, for both the USB and CPU divider blocks resulting in 48 MHz. The CCLK was then connected to the peripheral clock generator which controls the rate of the CCLK signal supplied to corresponding peripheral blocks (e.g., timers, UARTs, etc.).



**Figure 5-3: Clock generation diagram for the LPC2300 series (adapted from [51])**

The Keil RTX real-time kernel was configured to run up to 15 concurrent tasks, allowing a stack size of up to 1,200 bytes per task. The system timer utilized by the kernel was the peripheral timer 3 on the LPC2368 microcontroller and it operates at a frequency of 15 MHz. A tick value of 100  $\mu$ s was used by the kernel to control timing and scheduling of tasks and processes. This tick value was chosen as it provided sufficient granularity for the full range of timing functions required by the various tasks. The kernel utilized preemptive round-robin task switching multitasking for a total execution time of up to 0.5 ms per task (5 kernel ticks).

## **6 PC/hand-held Device Interface**

This chapter describes the design and implementation of the various components which made up the interface between the PC/hand-held device and the stimulator. First, the physical communication medium is discussed. This includes analysis of the requirements associated with this interface, which leads into the decision on the specific communication protocol. Then, the design and implementation associated with each feature supported by this interface will be described. These features included the ability to: a) transfer a stimulation protocol to and from the stimulator; b) acquire specific data (e.g., stimulator status, sensor data, stimulation parameters, electrode status); c) control ability to start, stop, pause and resume stimulation; d) adjust stimulation parameters; and e) modify the stimulation protocol.

### **6.1 Communication Protocol Selection**

The chosen communication protocol utilized in the design and implementation of the PC/hand-held device and stimulator interface was the USB 2.0 standard. This section will describe the assessment process used in deciding on the communication protocol and technology most appropriate for this interface. The procedure started by analyzing the applicable requirements composed from the requirements elicitation stage of the project. Then the evaluation of various communication technologies and protocols was carried out and compared for the final decision.

Function Type	Function
Control	Start stimulation
	Stop stimulation
	Pause stimulation
	Resume stimulation
Stimulation protocol	Download stimulation protocol
	Upload stimulation protocol
	Adjust stimulation parameters
	Modify stimulation protocol
Data acquisition	Get current stimulator status
	Get current stimulation parameters
	Get current electrode status
	Acquire input sensor data
	Get stimulator run and idle time

**Table 6-1: PC/hand-held device interface functionality**

From the requirement elicitation analysis described in Section 4.1, the PC/hand-held device to stimulator interface would be intended for a wide range of functionality, Table 6-1 above provides a sample list of functionality that would be provided by this interface. The types of functions performed through this interface will have different requirements in terms of size, speed of operation, reliability and the frequency that they are performed. Table 6-2 below provides a high-level approximate performance overview of the specific functions supported by the PC/hand-held device to stimulator interface. The parameters in this table are based on the overall system requirements and functionality and serve as a starting point for assessing different communication protocols.

Function Type	Function request size	Speed	Reliability	Frequency
Control	Small (~ Bytes)	High (ms)	High	Low - Med
Stimulation protocol	Medium (~ KB)	Low (~s)	Medium	Low
Data Acquisition	Small-large (B-MB)	Medium (ms-s)	High	Low - High

**Table 6-2: PC/hand-held device to stimulator interface general function requirements**

Examining the overall results from this table, the required communication interface should support a bandwidth around 50 KB/s (assuming 50 Byte transfer over 1 ms), and have low latency (ms) for fast responses and high reliability (minimum number of required re-transmits and error correction). Aside from the performance capabilities, power requirements were also considered. Since the stimulator would run from battery power, important energy saving consideration had to be taken into account for maximizing battery life when deciding on the type of communication technology in the design.

Standard	Medium	Bandwidth	Latency	Power	Range (meters)
Serial (RS-232) [52]	Wired	Up to 14.4 KB/s	~2 ms	~ 125 mW	Up to 1800
USB 2.0 [53-54]	Wired	Up to 60 MB/s	~0.1 ms	Up to 2.5 W (Host)	Up to 5
Bluetooth [55-56]	Wireless	Up to 375 KB/s	~25 ms (10 m)	~ 2.5 mW (10m)	1-100
ZigBee [57]	Wireless	Up to 31 KB/s	~16 ms (10 m)	~ 0.5mw (10-100m)	Up to 1500

**Table 6-3:** Summary of common communication standards applicable to design

Various communication standards had been considered for this interface, Table 6-3 provides a summary comparison of the most applicable standards that were considered in the design. The overall decision criteria in which each standard was evaluated against was as follows: compatibility, performance and power consumption. The decision process was divided based on the type physical communication medium, namely wired and wireless communication standards. The best choice for each was made, and finally a decision was made between using the wired or wireless option for the solution.

Starting with the wireless communication standards, Table 6-4 provides an advantages and disadvantages comparison of the Bluetooth and ZigBee wireless standards. Bluetooth offers over about ten times the bandwidth capacity compared to ZigBee, but ZigBee has lower power consumption per meter thus longer range (in relation to transmit power). The latency between these two wireless standards was comparable, with ZigBee being on average 9ms lower. Because of the specific requirements and capabilities of communicating between the stimulator and the PC/hand-held device, having the ability to communicate over a range of hundreds of meters was not a requirement since these components would most likely be within the same room. In addition, Bluetooth was intended as a cable replacement technology, and has been successfully used in many commercial applications for this purpose. Therefore the technology of choice between these two wireless standards was Bluetooth because of these key advantages.

Standard	Advantages	Disadvantages
Bluetooth	Bandwidth	Range
	Cable replacement	Power consumption
	HW, SW Support	
ZigBee	Power Consumption	Bandwidth
	Range	Support

**Table 6-4:** Wireless communication standard comparison

The two wired standards which were considered for this interface were the USB 2.0 and the recommended standard 232 (RS-232), Table 6-5 below provides an advantages and disadvantages comparison of these standards. Both options use a serial communication stream between the source and target devices, in this case the stimulator and PC/hand-held device. Comparing these two standards, RS-232 came out on top in its simplicity to be incorporated into the interface design and in its ability to use longer cables when compared to USB 2.0. USB 2.0

on the other hand was advantageous in its bandwidth potential (up to 60 MB/s), latency, power management and overall compatibility with today's devices and PCs. These aspects were the most important criteria when it came to choosing the appropriate wired interface technology, and because of this reason, the USB 2.0 standard was the desired wired interface choice over the RS-232 standard.

Standard	Advantages	Disadvantages
USB 2.0	Compatibility	Cable length
	Bandwidth	
	Power management	
RS-232	Simple implementation	Compatibility
	Range	Bandwidth
		Power consumption

**Table 6-5:** Wired communication standard comparison

Both the Bluetooth and USB 2.0 standards were potential choices for use in the communication interface between the stimulator and PC/hand-held device. At this point the choice was narrowed down to choosing between a wired and wireless interface. The main advantage with the wireless interface (Bluetooth) was the portability aspect, requiring no cabling to the stimulator unit, and therefore more convenient to move and configure, being physically independent from other components of the overall system. The main advantage when using a wired interface (aside from performance), specifically USB 2.0, was its ability for the host device (in this case the PC/hand-held device) to provide power to its client(s) (in this case the stimulator) which would offer a convenient way to charge the unit as well as communicate through one physical connection. Furthermore, USB 2.0 offered a higher communication reliability (less data corruption within medium – therefore higher throughput), higher bandwidth



and lower latency when compared to the Bluetooth standard. Therefore the communication standard of choice for use within the current design of the PC/hand-held device to stimulator interface was USB 2.0.

## **6.2 Interface Configuration**

In this section the specific USB 2.0 configuration with the LPC2368 microcontroller is discussed. The low level hardware interface drivers are provided by the RealView® Real-Time Library (RL-ARM by Keil an ARM® company, USA). Specifically the USB 2.0 drivers provided device interfaces for common USB 2.0 device classes. These interfaces had default support in Windows 2000, XP, Vista and 7 host PC/devices. The USB 2.0 port on the LPC2368 ARM 7 microcontroller was interfaced from a PC (host – Windows 7) to the stimulator (client – LPC2368 ARM 7) through the USB Communications Device Class (USB-CDC).

The USB-CDC device class specification did not attempt to dictate how all communication equipment should use the USB 2.0 interface, but rather to define an architecture that was capable of supporting any communication device [54]. It describes a framework of USB 2.0 interfaces, data structures and requests where a wide range of communication devices can be defined and implemented. To this effect, the USB-CDC was configured to operate as a virtual serial connection from the host (PC) through the UART USB driver. This configuration through USB 2.0 had been selected in order to provide the greatest flexibility in the communication design, including backward compatibility from the PC side to allow the incorporation of the existing stimulation protocol user software from Compex Motion (used to download/upload stimulation protocols) with minimal alterations.

### 6.3 Received Data Processing

The overall received data handling design from the USB interface included two main tasks. The first task read incoming data from the USB hardware buffer and wrote it to a local memory buffer ready for further processing (USB driver task), and the second task read the data from this buffer and processed it to handle various services and requests with the rest of the stimulator control system (USB data processing task). The USB driver task implemented the USB-CDC device class from the USB 2.0 standard, the specific implementation (driver) for the ARM 7 platform was provided by the Keil ARM library as an open source implementation. The USB data processing task had been designed and developed to work in conjunction with the USB driver task, and it is found in the “pcControl” class. In this section, the design and implementation of the USB data processing task will be discussed.

The USB data processing task continuously checked for new data at the local USB memory buffer, if a new data element was received it was extracted for further processing. The size of a single data element was one byte. Assuming there was no current running request through the USB 2.0 interface, this task waited for the arrival of a valid operation code sequence (series of bytes) from the PC/hand-held device. Communication was established when the controller received a pre-determined operation code. Each operation code was a sequence of bytes, which was used to determine the specific function or request intended for the controller subsystem. The functions corresponding to each operation code are listed in Table 6-6 below.

USB operation code functions	
22 – Byte sequence	Legacy stimulation protocol receive
22 – Byte sequence	Legacy stimulation protocol transmit
0x11, 0xB9, 0x46, 0x54, 0x52, 0xEC	Stimulation protocol transmit
0x11, 0xB9, 0x50, 0x43, 0x52, 0xE5	Parameter change
0x11, 0xB9, 0x47, 0x50, 0x52, 0xE9	Get parameter
0x11, 0xB9, 0x53, 0x43, 0x52, 0xE8	Change stimulation protocol
0x11, 0xB9, 0x53, 0x53, 0x52, 0xF8	Stop stimulation
0x11, 0xB9, 0x53, 0x50, 0x52, 0xF8	Pause stimulation
0x11, 0xB9, 0x53, 0x42, 0x52, 0xE7	Resume stimulation

**Table 6-6: USB operation code functions**

The general format of each operation code (except legacy codes due to compatibility requirements) was 6 bytes in length. The first two bytes were used as a header and they were equal for all operation codes. The next three bytes were used to identify the specific request function and finally the last byte was used as a checksum for the entire operation code received as a whole. The checksum was used as a means of verifying the integrity of the received operation code.

## 6.4 Stimulator Function Requests

Once the controller verified the function requested from the received operation code, the USB data processing task continued and served the request. While the request was being processed, no other requests were handled until the current one was completed or “times out” in the event of a disconnection. Therefore the request handling scheme used was sequential, where one request was fully processed before the next operation code was received or serviced. The reasons for using a sequential scheme vs. a parallel one was due to several factors primarily related to performance on the microcontroller side. If a parallel request handling scheme was used, the time to process each request would increase reducing the speed of handling each request.

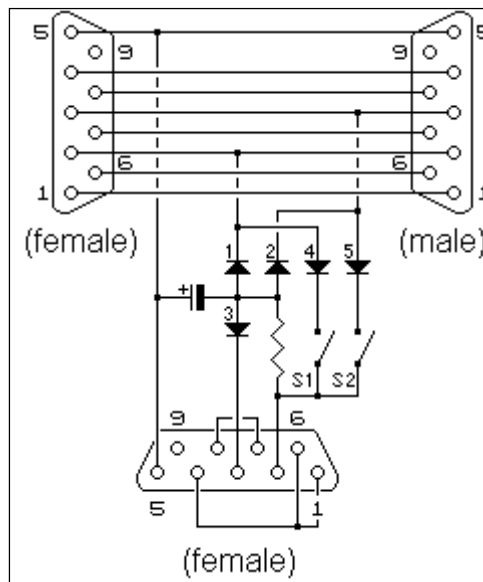
Another reason for this decision was for keeping the interface communication straight forward and simple with minimized overhead, this would make the process of adding further support more streamlined in the future. In the subsections below, specific design information is provided regarding each function request currently supported by the controller's USB interface. All validation and verification testing regarding each function request can be found in the experimental results chapter (Section 10.2).

### **6.4.1 Bidirectional Stimulation Protocol Transfer**

The stimulation protocol transfer requests included the download of a stimulation protocol to the controller (stimulator) from the PC and the upload of a loaded stimulation protocol from the controller to the PC. The PC software used to create custom stimulation protocols for the new generation stimulator was the one which had been developed for the Compex Motion stimulator. The reasons for using this software was primarily because the protocol encoding scheme developed for the Compex Motion stimulator would serve as a starting point for the base protocol in the new generation stimulator.

Therefore in the interest of time and familiarity for previous Compex Motion users, the legacy PC software was used to design, develop, manage and transfer stimulation protocols to and from the new controller. The main advantage included backwards compatibility to the previous generation software, which allowed users to utilize the vast number of proven stimulation protocols which had been created and applied for a various number of neuroprosthetic applications. This enabled current users to seamlessly transition to the new generation system without disrupting their current practices or processes and at the same time allowing new users a jump start in the development of custom stimulation protocols.

The first step in the design process of integrating the backward compatibility support to the PC/hand-held device interface was to mimic the communication protocol used in the Compex Motion PC to stimulator. The communication stream between the Compex Motion stimulator and its PC software was intercepted using a serial sniffer [58] with the schematic shown in Figure 6-1 below. The basic operation of this serial sniffer was to monitor the receive and transmit pins (pins 2 and 3 on a D-Subminiature 9 (DB-9) connector respectively). Sample stimulation protocols were transferred to and from the stimulator, and switches S1, S2 (Figure 6-1) were used to isolate the direction of data capture by the sniffer. This information was then analyzed and compared to decipher the specific communication protocol. The legacy features of the new controller were then integrated within the new controller subsystem in order to successfully communicate with the legacy PC software from the Compex Motion system.



**Figure 6-1: RS-232 serial sniffer schematic (Adopted from [58])**

The stimulation protocol created by the Compex Motion PC software was rendered into a 2 KB file size regardless of stimulation type or length; refer to Section 9.2 for more details regarding

the stimulation protocol data file encodings. The legacy communication scheme (in transferring a stimulation protocol to/from the stimulator) included a 7 byte synchronization message after every 240 bytes of transferred data (stimulation protocol data file). The synchronization message contained a 2 byte checksum for the received 240 byte block, a constant character, a count up byte, a countdown byte, and the last two bytes held information regarding the type of data in the received block.

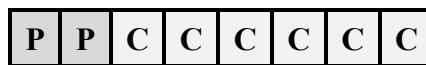
The encoded stimulation protocol file which was received by the controller also contained an additional 2-byte checksum after every 254 bytes. Therefore within the 2-KB stimulation protocol files there were 8 - 256 byte blocks of data including the corresponding checksums. Once a stimulation protocol file was received from the Compex Motion PC software, the 8 checksums were then verified for integrity. The two byte check sums were calculated by the addition of each byte within the block and restricted to a 2-byte result. The use of a 16-bit check sum was able to detect all single bit errors and all error bursts of 16 bits or less, it would also detect 99.998% of longer error bursts [59]. If an error was found in the received stimulation protocol the controller displayed the message “StimProt DL FAIL” and the received data was discarded.

### **6.4.2 Stimulation Parameter Change**

When a stimulation protocol was developed by a HCP for a specific patient it was often the case that the most appropriate stimulation intensities were not known initially or changed over time. It was therefore advantageous to allow the HCP the ability to adjust these intensities for specific channels without having to re-download the stimulation protocol multiple times. As described in Section 2.3.2.2, the intensity of stimulation was controlled through the amplitude and or the PW of the stimuli signal.

On the Compex Motion stimulator system, the amplitude stimulation parameter was adjustable directly from the stimulator unit itself. This required additional buttons on the stimulator unit, which cluttered and unnecessarily complicated its packaging. On the new generation stimulator, the interface provided on the stimulator unit directly was to simply allow starting and stopping of a stimulation sequence (primarily for safety purposes). All additional control capabilities were provided through the use of a PC/hand-held device which included the adjustment of stimulation intensities.

Once the parameter change operation code was received and verified at the USB interface, the controller expected an additional byte which held information pertaining to which stimulation parameter was requested to change and for which channel. Figure 6-2 below shows the bit composition of this parameter identification byte. As shown, the first two bits labelled 'P' hold information regarding which parameter was being requested to change and the last 6 bits identify the specific stimulation channel. Table 6-7 below provides the bit description mapping for the parameter bits and the channel bits.



**Figure 6-2: Bit composition of the parameter change identification byte**

Parameter	P P	Channel	C C C C C C
Amplitude	0 0	Channel 1	0 0 0 0 0 0
PW	0 1	Channel 2	0 0 0 0 0 1
Frequency	1 0	Channel 3	0 0 0 0 1 0
Not used	1 1	Channel 4	0 0 0 0 1 1

**Table 6-7: Parameter change identification bits**

Once the parameter change identification byte was received, the next two bytes held the new parameter data. The last byte in the parameter change request was a check sum byte used to verify the integrity of the data received and was processed as detailed in Section 6.4.1. The new parameter data was then loaded in the appropriate parameter change registers which were set on the next update of the output stage parameters (see section 9.3 for stimulation decoding and execution design and implementation).

### **6.4.3 Stimulation Parameter Retrieve**

Similar to the parameter change requests, the parameter retrieve requests provide the current value of the requested parameter to the PC/hand-held device. The use of this function allowed users to observe the values of different stimulation parameters during stimulation. The data retrieved using this function could be used by the PC/hand-held device to generate a visual display of the stimulation parameters during a stimulation sequence.

After the “get parameter” operation code was received and verified, an additional byte was received detailing which parameter and for which channel the request was for. This byte had the same composition and bit layout as the parameter change identification byte explained in Section 6.4.2. Once this byte was received and decoded, the appropriate parameter information was acquired and packaged by the controller and sent through the USB interface to the PC/hand-held device. The return information was packaged as a four byte message. The first byte was used as header information indicating that the next three bytes contain parameter information data. The next byte was the parameter and channel identification byte and the next two bytes held the parameter information pertaining to the requested parameter. The requested parameter information was then loaded from the appropriate parameter change registers which was last loaded onto the output stage.



#### 6.4.4 Stimulation Protocol Change

The stimulation protocol change capability enabled users to make changes in specific sections of the loaded stimulation protocol without re-downloading the entire stimulation program. The idea was that when the HCP worked on a stimulation protocol for a specific patient he/she would download the protocol to the stimulator multiple times until the desired operation was achieved. In these cases the user makes slight adjustments to various parameters and control switches in the base protocol but requires downloading the entire stimulation program to get the desired changes. This feature would allow the GUI software on the PC/hand-held device to provide the stimulator differential changes in the stimulation program compared to the original program already loaded. The benefits were an increased efficiency when stimulation program adjustments were made as there was considerably less data being transferred each time allowing for faster protocol development times.

This operation was only possible when the stimulator was in an idle state with the stimulation protocol loaded. The current design did not support protocol changes while a stimulation protocol was already running or paused due to undesired complications such as protocol corruption and unnecessary complexity. Once the stimulation protocol change operation code was verified, the first four bytes indicated the start and end indexes within the loaded stimulation protocol in which the following data would replace. The fifth byte was a checksum byte for the first four received bytes. At this point the expected number of bytes for the appropriate protocol change was calculated ( $n = \text{end index} - \text{start index}$ ). The next 'n' bytes were then received followed by a two byte checksum for this data block. Once the received data was verified (with checksum calculation), the appropriate changes were made to the stimulation protocol stored within the stimulator.

### **6.4.5 Stimulator Control through USB Interface**

The stimulation control component of the PC/hand-held device interface allowed users to start, stop, pause and resume stimulation. The stimulator unit included one button on its interface to start and stop stimulation primarily for safety purposes, the additional control was provided through the PC/hand-held device interface. The pause and resume feature allowed the HCP to temporarily stop stimulation programs to adjust parameters or provide a break to the patient during a treatment session.

The stimulator control command codes were encoded within the operation codes corresponding to each request type (e.g., start, stop, pause and resume). Within the fixed 6-byte operation code the 3<sup>rd</sup> 4<sup>th</sup> and 5<sup>th</sup> byte distinguish the control command. Once a control command was received by the controller it was verified to ensure the command was valid, for example if it received a start command when the stimulator was already executing a stimulation protocol or a stop command when the stimulation was already stopped. Once verified, the appropriate action was performed and the status and display were updated to reflect this change.

## **7 User Interfaces**

This chapter describes the design and implementation of the user interface components of the stimulator system. Both the user interface on the stimulator itself and the user interfaces on the PC will be covered. The stimulator's user interface encompassed an output display used for various status and error messages and an input interface to directly start or stop and turn on or off the stimulator. The PC component of the user interface included a software program to create and download/upload stimulation protocols to/from the stimulator and another software program to control the stimulator and view current status/parameter information.

### **7.1 Stimulator Interfaces**

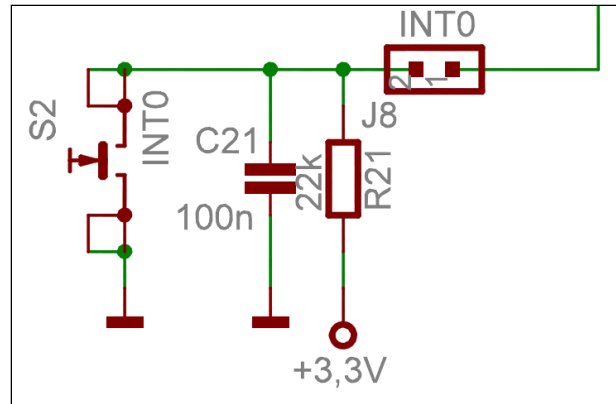
The user interface on the stimulator unit was intended to provide direct and easy access to the stimulator. The users accessing this interface would range from the HCP to patients to scientists and technicians. Therefore the information provided must be suitable for the wide range of users that would be accessing it. The interface was divided into two components based on input to the stimulator and output to the user. The input to the stimulator was comprised of an interface to power on and off the unit and another interface utilized to start and stop stimulation. The output to the user was comprised of a visual display where information would be instantaneously relayed back to the user.

#### **7.1.1 Stimulator Input Interface**

During the development of the stimulator's controller subsystem, the power to the microcontroller and peripherals was supplied through the USB 2.0 interface. Since this

connection was already used to communicate to with a PC, no extra interfaces were required to power the unit. Therefore the current interface to power up and power down the controller was controlled by the physical USB cable connection. Once the interface between the controller and output stage is finalized and final packaging of the stimulator is developed, an appropriate battery interface with power switch would be incorporated (see Section 11.2 for areas of extensions and future work).

The interface to start and stop stimulation directly from the stimulator was provided as a safety feature in the case that the user must immediately stop stimulation. Although this feature could also be accessed from the PC/Hand-held device interface as well, the PC/hand-held device does not provide hard real-time input to the stimulator and the stimulator's operation was designed to be independent to any external devices. Therefore, this dual-purpose (start/stop) input to the stimulator was interfaced using a single pushbutton which was directly connected the external interrupt 0 (EINT0, pin 53) of the LPC2368 microcontroller. The EINT0 was configured to interrupt the microcontroller on a falling edge (press of push button) of the input signal. Figure 7-1 below illustrates the interface of the pushbutton switch to the EINT0 of the microcontroller. The 100 nF capacitor and the 22 k $\Omega$  resistor provided a filter to debounce the switch in order to filter out false falling edge detections by the microcontroller. This input switch interface was provided on the MCB2300 evaluation board.



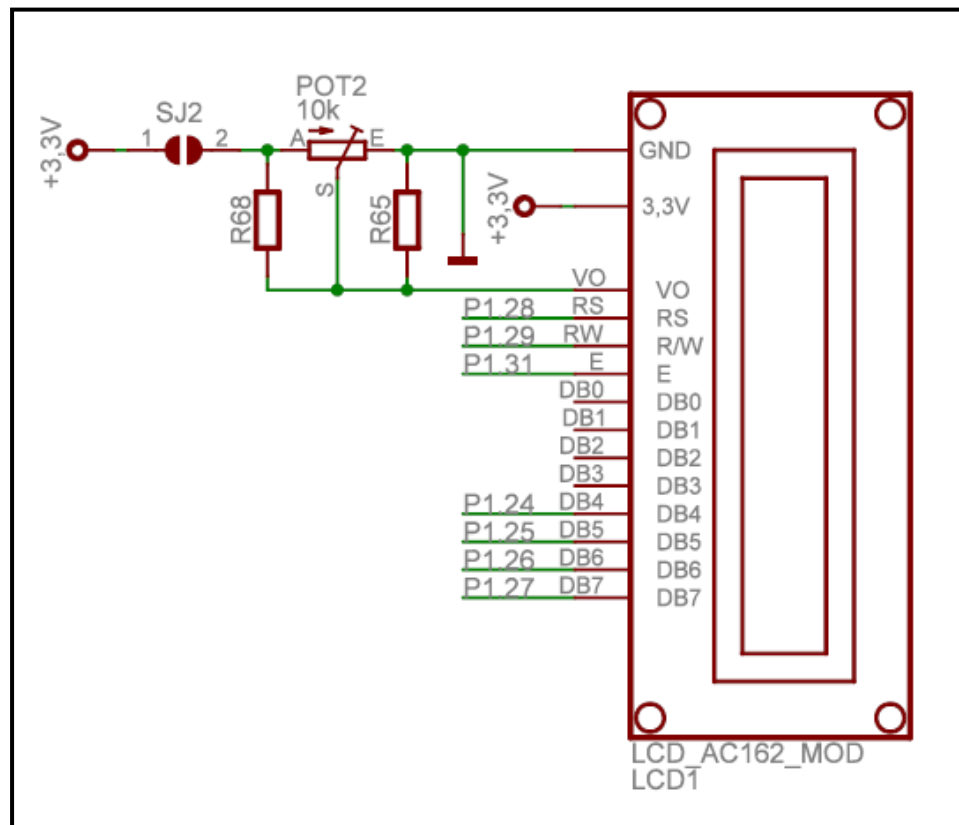
**Figure 7-1: External interrupt 0 switch input diagram (adopted from [60])**

The “ProjectMain” class processed user input requests directly since these requests affected the operation of the entire stimulator. Once the push button was pressed, an external interrupt was generated (within the LPC2368 microcontroller) which triggered the corresponding interrupt service routine (ISR) for this input. The ISR acquired the current state of the stimulation (e.g. stimulation on, stopped, or paused), and based on this state it performed the appropriate action. For example, if a stimulation protocol was being executed, the ISR initiated an immediate stop of all current stimulation and brought all pulse amplitudes to zero. If the stimulator was in idle state with a valid stimulation protocol loaded, the ISR initiates the start of the stimulation decoding and execution procedure.

### 7.1.2 Stimulator Output Interface

The stimulator’s output interface was intended to provide the user with information about the state of the stimulator including any error messages. It also provided a means to relay messages about the stimulation to the patient, for example custom messages directing the patient about what to expect next or to perform a specific action in order to trigger a desired stimulation pattern. This functionality was implemented in the “displayControl” class. Since the messages

on this display were intended to be straight forward and simple, the display size used in the Compex Motion stimulator was sufficient. Therefore the display interfaced on the stimulator unit was a two-line 16 character liquid crystal display (LCD) connected using a 4-bit interface to the LPC2368 microcontroller. This LCD display interface was provided on the MCB2300 evaluation board and its schematic is shown in Figure 7-2 below.



**Figure 7-2: LCD display interface to LPC2368 (adopted from [60])**

The LCD display on the stimulator relayed status and error messages in a static manner until another message was shown. For example when the stimulator was turned on, the message “Stimulator Idle” was shown and would remain on the display until the next event was triggered. Each status and error messages were given ID numbers to facilitate referencing by administrative/technical users. Other messages originating from the stimulation protocol and

intended directly for the patient were designed to be displayed for a custom length of time (encoded in seconds) which is programmed through the stimulation protocol. Once the allotted time had elapsed, the previous status message was displayed again. For example if the message “press button” was displayed for 5 seconds during a stimulation sequence, once the time was up the message “StimP Running!” (stimulation protocol is running) was displayed again – indicating that a stimulation sequence was still running. Table 7-1 and Table 7-2 below provide a comprehensive list of all the status and error messages currently supported by the controller subsystem.

<b>Status Messages</b>	
<b>Message</b>	<b>Description</b>
STATS 001: Stimulator Idle	No active stimulation
STATS 002: StimP Running!	Stimulation protocol is running
STATS 003: StimP Paused!	Stimulation protocol is paused
STATS 004: USB-Leg FRC conE	USB legacy file receive connection established
STATS 005: USB-Leg FRC OK!	USB legacy file receive connection completed
STATS 006: USB-Leg FTC conE	USB legacy file transmit connection established
STATS 007: USB-Leg FTC OK!	USB legacy file transmit connection completed
STATS 008: USB-FTC conE	USB file transmit connection established
STATS 009: USB-STIMC conE	USB stimulation protocol change request
STATS 010: USB-STIMC OK!	USB stimulation protocol change completed
STATS 011: USB-PARAMC conE	USB parameter change request
STATS 012: USB-PARAM OK!	USB parameter change completed

**Table 7-1: Comprehensive list of status messages managed by the controller subsystem**

<b>Error Messages</b>	
<b>Message</b>	<b>Description</b>
ERROR 001: USB-Buf OverFlow	USB circular buffer overflow has occurred
ERROR 002: USB RQST TIMEOUT	Current active USB request timed out
ERROR 003: StimProt DL Fail	Stimulation protocol download failed (corrupted)
ERROR 004: No Stim Prog	Start stimulation command issued when no stimulation protocol is present
ERROR 005: USB StimC Fail	Stimulation protocol change request failed
ERROR 006: USB ParamC Fail	Parameter change request failed
ERROR 007: Timed Trig Fail	Stimulation real-time trigger failure detected.

**Table 7-2: Comprehensive list of error messages managed by the controller subsystem**

## 7.2 PC Interfaces

The user interfaces provided on the PC served two purposes; to allow HCP, therapists or clinician to design and develop custom stimulation protocols and, to control and retrieve status information from the stimulator. Access to the stimulator from these interfaces provided the user with full functionality and capabilities available on the stimulator's controller. For example, all functionality described in Chapter 6 can be accessed and utilized from these interfaces alone making it a powerful and convenient package for those administering FES.

### 7.2.1 Stimulation Protocol Interface

The stimulation protocol component of the PC's interface provided the user (HCP, therapist or clinician) the tools to design, develop and manage stimulation protocols for specific patients, with the ability to download and upload these protocols to/from the stimulator. As detailed in Section 6.4.1, the software utilized for this functionality was initially made for the Complex Motion stimulator system (entitled - Program Complex Motion) and thus the new stimulator controller subsystem provided backward compatibility to this software. The main interface



window displayed when the “Program Complex Motion” was executed is shown in Figure 7-3 below.



**Figure 7-3: Program Complex Motion primary window**

The red square contains the programming functions associated with stimulation channel 2, in this area one can select the default amplitude and set the maximum amplitude (in mA) and PW for this channel (in  $\mu$ s). The timeline section of this area is where various functions can be selected to create the stimulation sequence for this specific channel. These functions are added by selecting the desired object in the “End” entitled boxes. These functions are called primitives and further details regarding their operation are described in Section 9.1.

The green square provides the user with further stimulation configuration details. The “User Information” button allows the user to provide textual information regarding the specific stimulation protocol such as treatment information, and subject details (i.e., name, ID, date of birth). The “Pulse Width (PW)” button allows PW ramps to be defined for each stimulation channel. The “Amplitude + Freq” button allows the user to configure amplitude values for each stimulation channel and stimulation frequencies. Buttons “Analog Amp Control” and “User Interactions” are used to configure input sensor interactions and control data. Finally, the “Text” button is used to pre-set textual information intended to be displayed to the patient through the LCD interface described above.

In the blue square the user can configure specific stimulation characteristics such as type and control of stimulation waveforms and timeout information. In the yellow square the default stimulation frequency can be set, the “Read Chip Card” button will read the current active stimulation protocol program on the stimulator and load it into the program. The “Program Chip Card” button is used to download the current stimulation protocol to the stimulator and the “Save” button allows the user to archive the current stimulation protocol on his/her PC. Loading saved stimulation protocols is possible when the Complex Motion software starts up or after saving the current stimulation protocol.

### **7.2.2 Stimulator Control Interface**

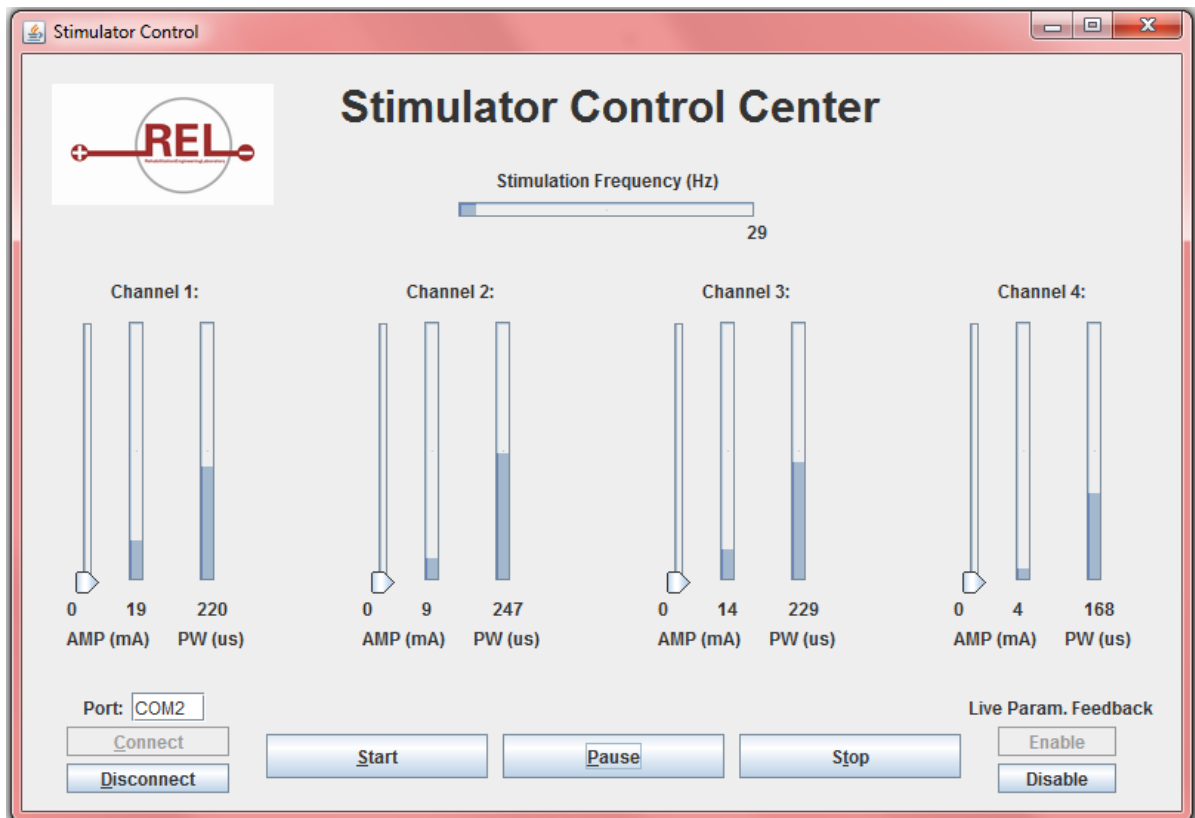
The stimulator control software was a program which provided the user with a powerful and easy to use graphical control interface (GUI) over a stimulation unit. The user was able to start, stop, pause and resume stimulation at any point by the click of a button. Data regarding current stimulation parameters was also gathered and graphically displayed for the user, allowing a

visual representation of the immediate stimulation being administered. Stimulation amplitude values for specific channels could also be changed through this interface while the stimulation is running or when paused.

The stimulator's control GUI was developed using the Java programming language with the Windows 7 operating system. The graphical interface component of the software was designed using the javax.swing package which contained the necessary classes to generate the required graphical components such as windows, frames, buttons and labels. The stimulator's PC/Hand-held device interface was used by this software as the communication pathway to the stimulator's controller (through USB 2.0). This interface was connected using a virtual RS-232 serial port in Windows 7 through the USB-CDC standard (see Section 6.2). The stimulator's control software (host pc) interprets this physical USB connection as a RS-232 serial port interface to the stimulator (client device). The serial communications administered with Java was executed through the communications application programming interface (API) classes (also known as javax.comm) which provided the control software direct access to the RS-232 serial ports in Windows. Some of the main features provided by this API included port mapping, port configuration (baud rate, stop bits, parity), access to specific RS-232 line signals and transfer of data over RS-232 serial ports. The stimulator control software relied on this communication pathway to perform all control commands and data retrieval to/from the stimulator. The stimulation control software used the communication standard described in sections 6.3 and 6.4 over the USB 2.0 interface connection to interact with the stimulator's controller subsystem.

The overall design goal of the stimulator control software was to allow the user direct interaction with the stimulator in a simplified and effective manner. Figure 7-4 below displays a screenshot

of the control software while running on a Windows 7 PC. The GUI provided an intuitive layout which could be understood and utilized by a wide range of users. The stimulation frequency which was common among all stimulation channels was displayed on the top center of the window with a horizontal graphical bar and numerical value in Hz shown on the right. The pulse amplitude and PW information was grouped by stimulation channel. Similar to the stimulation frequency, this information was displayed using vertical graphical bars and respective numerical values shown under each. On the left of each stimulation channel grouping there was a slider bar which was used to modify the stimulation amplitude (in mA) for a specific stimulation channel. This was done by sliding the bar to the desired level with the respective numerical value displayed under the slider.



**Figure 7-4: Stimulator's control center PC software**

At the bottom left of the interface, the text box labeled “Port:” and the two buttons named “Connect” and “Disconnect” were used to configure and connect/disconnect the communications connection to the stimulator. As described above, this connection was established through a virtual serial port over USB, therefore the communication (COM) port number (i.e., COM2) to the appropriate virtual COM port had to be provided. Once the connection to the COM port had been established the “Connect” button was disabled and the “Disconnect” button was enabled, this provided the user with feedback regarding the status of the COM port connection. The next three buttons at the button named “Start”, “Pause” and “Stop” were used to directly command the stimulator to perform these respective actions on the stimulator. The “Live Param. Feedback” buttons were used to either enable or disable the acquisition and display of stimulation parameter data. When enabled it provided the user with visual and numerical information on the stimulation parameters for each channel as the stimulation was being executed, which can be seen in Figure 7-4 above, which was captured during the execution of a stimulation sequence. The “\_” character under a specific letter on each button indicates the keyboard shortcut keys to activate the specific button by selecting “Alt + X” where X is replaced with the underlined character.

## 8 Input Sensor Interface

This chapter describes the design and implementation details pertaining to the input sensor interface to the stimulator's controller subsystem. This interface provided the capability to connect various sensors or sensor systems to the stimulator, which could be used to trigger stimulation sequences, regulate stimulation amplitudes in real-time and acquire sensor data. The discussion starts by providing some example scenarios where sensors can be used in real-world FES applications. Specific types of sensors the controller would interface to are analyzed with a summary of their physical characteristics (digital or analog type input) and applications. This information was then used to define the physical requirements of the input interface. Finally the design and implementation with the LPC2368 microcontroller is provided.

### 8.1 Application

When FES therapy is administered for rehabilitation purposes, input control over the stimulation patterns is often required. The input is usually initiated by the patient through the use of one or more input devices or sensors. Based on the type of input received by the stimulator, various pre-defined actions can be executed such as running another stimulation sequence, adjust stimulation intensity and to continue or stop stimulation. These input devices and sensors can be conveniently mounted on the user's assistive device or the user him/herself for increased accessibility and transparency. The integration of sensors within an FES system can also be used to measure a variety of biological signals such as force, muscle activity (EMG), joint position, angular velocity etc. These signals are typically used to automatically trigger or control stimulation patterns and parameters during the therapy [16].

An example application where an input sensor was required for a successful FES treatment was that of the drop foot neuroprosthesis. The typical drop foot stimulator stimulated the common peroneal nerve during the swing phase of gait. One common way to implement this stimulation function was with the use of a heel switch. The heel switch provided an input to the stimulator, indicating when the treated foot lost contact with the ground. The stimulator used this input as a trigger for stimulating the peroneal nerve; stimulating when the heel lost contact with the ground and stop stimulation when the heel contacted the ground again. In this case, the user controlled the timing of the stimulation based on his/her gait position.

## **8.2 Types of Sensors**

There were various types of sensors and sensor systems that could be integrated with the stimulator to develop the desired neuroprosthesis. These sensors can be divided into two categories being digital and analog type sensors. Analog sensors are those that can represent many different values or levels such as a sliding potentiometer. Digital sensors are those that represent two logical values, such as switches or push buttons that can be either activated or deactivated. Analog sensors can also be represented as digital sensors by using a switching value or level, for example if the value is greater than a pre-defined level the sensor can be interpreted as activated and if lower than that level it is deactivated. In general, the digital sensors were used as simple triggers to select between stimulation sequences and analog sensors were used for continuous regulation of stimulation intensities and to detect more complex control triggers from the user.

Sensor	Type	Application Example
Push button	Two states (digital)	Subject presses push button to initiate finger and thumb flexion, subject presses push button again to perform finger extension [25].
Slide Potentiometer	Variable resistor (analog)	Subject slides potentiometer in one direction for finger extension, subject slides potentiometer in opposite direction for finger and thumb flexion. Resistance of the potentiometer is used to define the grasping force generated by the neuroprosthesis [25].
Force sensor	Measures applied force (analog)	Gait phase recognition sensor constructed from three force-sensitive sensors, a gyroscope and a rule-based observer. Can identify four gait phases during walking which can be used by drop-foot neuroprosthesis for time-sensitive stimulation control [25].
Gyroscopes	Detect rotation (analog)	
Inclinometers	Measures tilt (analog)	Using a tilt sensor to detect thigh movement. The stimulus is turned on when the tilt signal rises above the ON threshold which corresponds to a predefined forward leg position. Stimulus will be turned off when tilt falls below a second threshold or a preset stimulation time [61].
Goniometers	Measures angles (analog)	Use of goniometers attached across the joins of the affected leg and pressure sensors installed in the insoles of subject's shoes to monitor gait for stimulation control [62].
EMG	Measures muscle activity (analog)	To measure voluntary muscle activity which can be used to control or trigger the neuroprosthesis [25].

**Table 8-1: Summary of applicable sensors with applications.**

Depending on the intended application of the neuroprosthesis, different sensors can be integrated in order to measure the desired physical values/phenomena. Table 8-1 above contains a list of some sensors and sensor systems that have been used in the past in different FES applications.



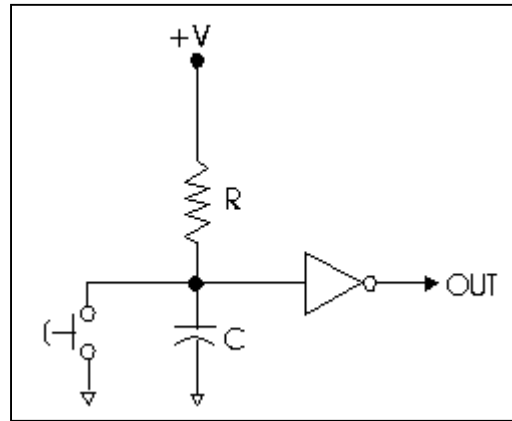
The information provided includes the type of data each sensor is intended to capture and an application example where the type of sensor can be used as part of a neuroprostheses.

### **8.3 Interface Design**

The various sensors and sensor systems would be interfaced to the LPC2368 ARM microcontroller. The microcontroller processes the input from the attached sensor(s) based on the pre-programmed stimulation protocol. The interface made use of two different components of the LPC2368, depending on the type of sensor input, i.e., a simple switch or button type (digital) or an analog sensor. Digital type sensors were interfaced through the external interrupt 1 input (EINT1 – pin 52) on the LPC2368. This input was configured as a falling edge-sensitive trigger which caused an external interrupt when a falling edge was detected. This input was configured to trigger on a falling edge because the input pin was kept at 3.3 V from the microcontroller by default. Therefore, for a basic switch integration circuitry which took this input to ground, was all that was required to integrate a simple pushbutton (without the use of any hardware debouncing).

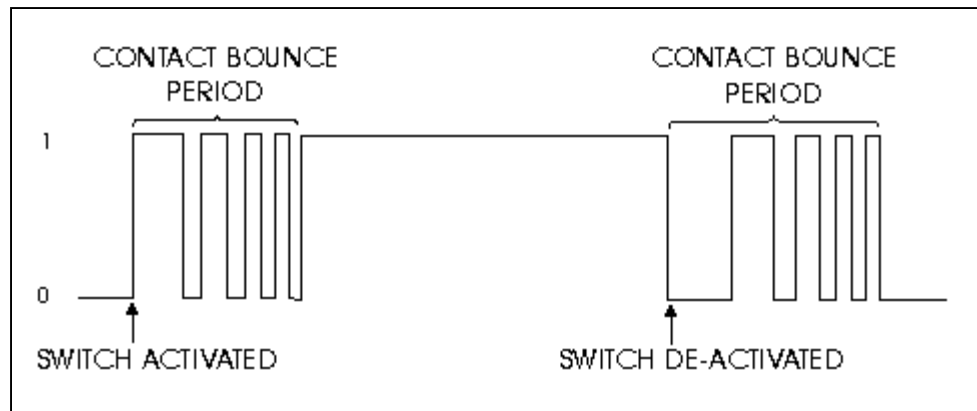
The digital input interface on the LPC2368 microcontroller did not provide any hardware debouncing circuitry and it was the responsibility of the neuroprosthesis designer to incorporate the appropriate hardware de-bouncing solution. The reasoning was that different types of digital input devices would require different components in the hardware de-bouncing circuit and it was therefore not practical to design a generic or adaptable hardware debouncing circuit as this would unnecessarily increase complexity and cost of the stimulator without providing critical functionality in return. A simple hardware de-bouncing circuit in the case of a push-button

switch is shown in Figure 8-1 below. It uses a resistor-capacitor (RC) time constant to filter out the contact bounce periods when the switch is activated and deactivated.



**Figure 8-1: Sample hardware de-bounce circuitry (adopted from [63])**

These bounce periods, as shown in Figure 8-2 below, can be interpreted as separate switch activation and deactivation periods by the microcontroller from a single switch event. The chosen RC value depends on the type of digital input device and its intended application. If the input device is a simple push button switch intended for use by the subject then 0.1 seconds is a recommended value for the resolution between switch activations. Finally, the use of the buffer is to provide a sharp switch transition (high to low or low to high). Since the low pass filter (LPF) slows the switching between low to high or high to low, this gradual switching can be interpreted as different logical levels by the microcontroller thereby re-introducing the debouncing issue.

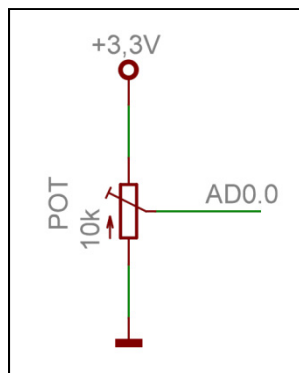


**Figure 8-2: Before proper hardware debouncing is performed (adopted from [63])**

Software debouncing has been incorporated where repeated switch transitions within a programmable time (currently set at 0.1 seconds) interval are ignored. The consequences of switch de-bouncing is that any valid transitions within this period are lost, therefore it is essential to choose the appropriate invalid transition time based on the intended use and input device.

The analog sensor interface was provided through the microcontroller's ADC inputs. On the LPC2368 microcontroller the ADC provided conversion of up to 6 multiplexed input pins with 10 bit accuracy. Each converter included a 4.5 MHz (max) clock, which was required by the successive approximation process (the conversion between analog inputs to their digital representation). A fully accurate conversion required 11 of these clocks, thereby providing a 10 bit conversion sampling rate of 409 KHz. The implementation utilized channel 0 of the ADC on the LPC2368 (AD0.0 – pin 9). The clock for the ADC was configured to 3 MHz which was the closest clock value to the 4.5 MHz maximum, based on the processor clock and configurable clock divisors, therefore providing an overall sampling rate of 272 KHz.

The analog sensor interfaced to the ADC port on the microcontroller was a turn dial potentiometer made by Iskra (model 10k-6288) and its interface on the MCB2300 evaluation board as shown in Figure 8-3 below. The ADC conversions were handled through the use of Interrupt Requests (IRQs) within the microcontroller. An IRQ was triggered for every finished conversion. The microcontroller processed this data and interpreted the incoming analog signal. This information was then used to compile the control signals from the subject as part of the neuroprosthesis. At this point one input interface was provided for digital sensors and one for analog sensors. This was sufficient to demonstrate its applicability and functionality as part of the overall stimulator system.



**Figure 8-3: Potentiometer sensor interface diagram to LPC2368 microcontroller (adopted from [60])**

## **9 Stimulation Protocol**

This chapter describes the design and implementation associated with the encoding, decoding and real-time execution of supported stimulation protocols. The encoding and packaging of stimulation protocols was performed on the PC side by the stimulation programming software, the decoding and real-time execution was performed by the controller subsystem within the stimulator unit. The overall chapter is divided into three logical parts starting with a discussion of the supported stimulation tasks and control functions, which are supported with an application based example. The design pertaining to the encoding procedure of stimulation protocols is then analyzed, showing how the stimulation tasks and control functions were efficiently compressed and packaged. Finally, the design and implementation of the decoding and execution of stimulation protocols with smart pre-unpacking and real-time execution to the output stage is detailed.

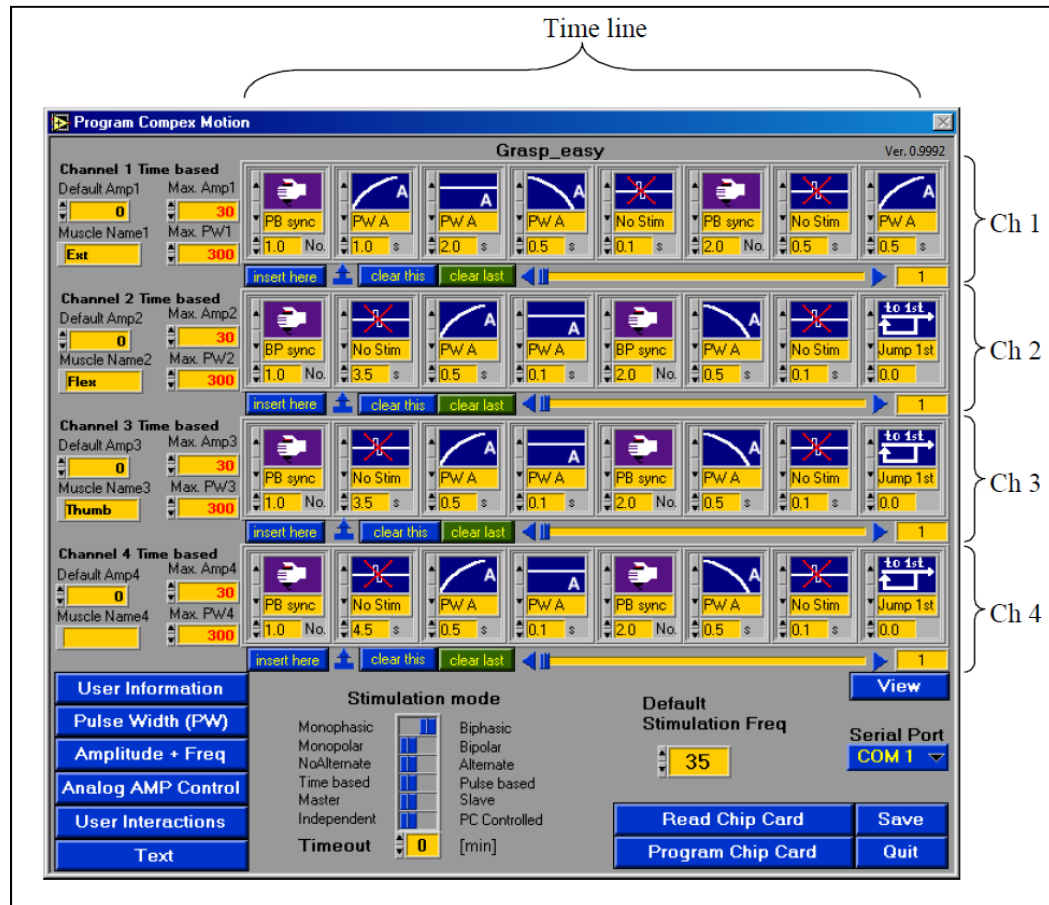
### **9.1 Supported Stimulation Tasks & Control**

The stimulation tasks and control functions are what make up the characteristics of the stimulation administered to the patient. The therapist or clinician utilizes various stimulation functions in order to implement a desired stimulation protocol. From the systems engineering point of view the main goal was to provide the necessary functional support to allow the formation of highly customized protocols intended for various FES applications. In this section a discussion of the supported stimulation tasks and control functions is provided, followed by how they can be integrated to create custom stimulation protocols through the use of a real-world example.

During the implementation of a custom stimulation protocol, the stimulation tasks and control functions were selected from the Compex Motion PC Software, these functions were called primitives. The HCP selects the appropriate primitives for each timeline when developing a custom neuroprosthesis. Each timeline corresponds to a specific stimulation channel, Figure 9-1 below displays the various primitives selected in each timeline (further details regarding the functionality of this user interface is provided in Section 7.2.1). Primitives were divided into three color-coded groups based on their effect on each stimulation channel. The blue set of primitives affected only the stimulation channel they resided in. The green set influenced all channels regardless of which channel they were placed in. The violet set influenced all channels through synchronizing the stimulation sequences and therefore had to be placed in all stimulation channels. The primitives were placed in consecutive order within a time line that described the chronological sequence of events that would be carried out by a specific stimulation channel.

The stimulation mode was also configured during the development of a custom protocol. In the stimulation mode section, one can configure how the stimulator operated and delivered the stimulation to the patient. The first three switches were used to configure the type of stimulation waveform utilized by the output stage, description and utilization of each type was provided in Section 2.3.1. The fourth switch was used to configure the stimulator to operate in time-based or pulse-based mode. In time based-based mode the stimulation pulse characteristics (amplitude, PW and frequency) were changed (refreshed) every 100ms. Within a 100ms window, the pulse characteristics remain constant. In pulse based mode, these characteristics were changed within the period of the stimulation frequency, thus providing the ability to change the stimulation characteristics after each pulse. The pulse-based mode was primarily used for research purposes, while the time-based mode was used for standard FES protocols and treatments. The last two










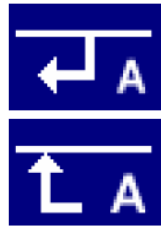
switches were not utilized in this version of the new stimulator as these features were not carried over.



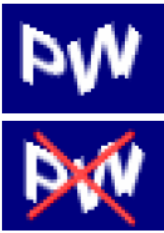





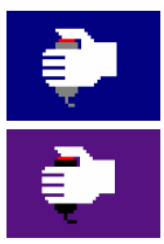

**Figure 9-1: Complex Motion main programming window**



The HCP implemented a stimulation protocol by configuring the desired stimulation mode and selecting the appropriate primitives based on intended functionality and placing them on each timeline corresponding to a stimulation channel. Table 9-1 below contains a summary of all 56 stimulation primitives supported by the stimulator. The HCP could choose and utilize any combination of these primitives in order to develop a custom stimulation protocol. Through the use of these various functions one could efficiently develop stimulation protocols intended for

simple and advanced neuroprostheses. To illustrate these capabilities a real-world FES example application which utilizes a subset of these primitives is provided below.

Primitive	Name (# primitives)	Description
	Turn off (1)	Stops all stimulation and turns off stimulator.
	End stimulation (1)	Terminates the stimulation sequence on specific channel.
	No stimulation (1)	Stops the stimulation on specific channel for a pre-defined amount of time.
	Delay (1)	Holds current stimulation parameters (from previous function) on specific channel for a pre-defined amount of time.
	Constant pulse (4)	Stimulates on specific channel at pre-defined constant PW 'A' for a pre-defined amount of time.
	Ramp up (2) / Ramp down (2)	Ramp up successively increases PW value for specified channel based on pre-defined ramp 'A' within pre-defined time interval. Ramp has the opposite function of ramp up.
	Amplitude change (4)	Changes the stimulation amplitude on specific channel to pre-defined amplitude 'A' within a pre-defined transition time.
	Frequency change (4)	Changes the stimulation frequency (entire stimulator) to pre-defined frequency 'A' within immediate interval.
	Jump to first (1)	Restarts the stimulation sequence for the selected channel by jumping back to the first primitive within its timeline.
	Marker (4) and Jump to marker (4)	The use of these two primitives provides the ability to configure a loop sequence within a specific channel. The "Mark" primitive (bottom) indicates the start of a loop sequence and the "Jump" primitive (top) indicates a jump back to the corresponding "Mark" primitive with a pre-define number of loops or infinite






		number of loops.
	PW randomization on (1) / off (1)	Activates (top) PW randomization for specific channel with a uniform probability distribution function within a pre-defined % range of the nominal value. Deactivates (bottom) PW randomization for specific channel.
	Amplitude randomization on (1) / off (1)	Activates (top) amplitude randomization for specific channel with a uniform probability distribution function within a pre-defined % range of the nominal value. Deactivates (bottom) amplitude randomization for specific channel.
	Frequency randomization on (1) / off (1)	Activates (top) stimulation frequency randomization with a uniform probability distribution function within a pre-defined % range of the nominal value. Deactivates (bottom) stimulation frequency randomization.
	Display text (4)	Displays textual information on the stimulator's display for a pre-defined amount of time.
	Play sound (2)	Plays audible sound sequence when primitive is executed.
	Synchronization (1)	Synchronizes all stimulation channels by waiting for all channels to reach this primitive within their timeline. Once all channels have reached this primitive, all channels will continue with the execution of proceeding primitives.
	Push button (1) / synchronized push button (1)	Push button (top) primitive provides user control of the execution on the specific channel. Once the specific channel reaches this primitive, it does not continue execution on this channel until the stimulator receives the push-button signal. The synchronized push button (bottom) primitive ensures all channels have reached this primitive before the stimulator will accept the push-button signal to continue execution.
	TTL (Transistor-Transistor Logic) trigger (1)	TTL primitive synchronizes all channels, once all channels have reached this primitive the stimulation is stopped and the stimulator waits for a negative slope trigger signal. Provides fastest trigger response from external devices.

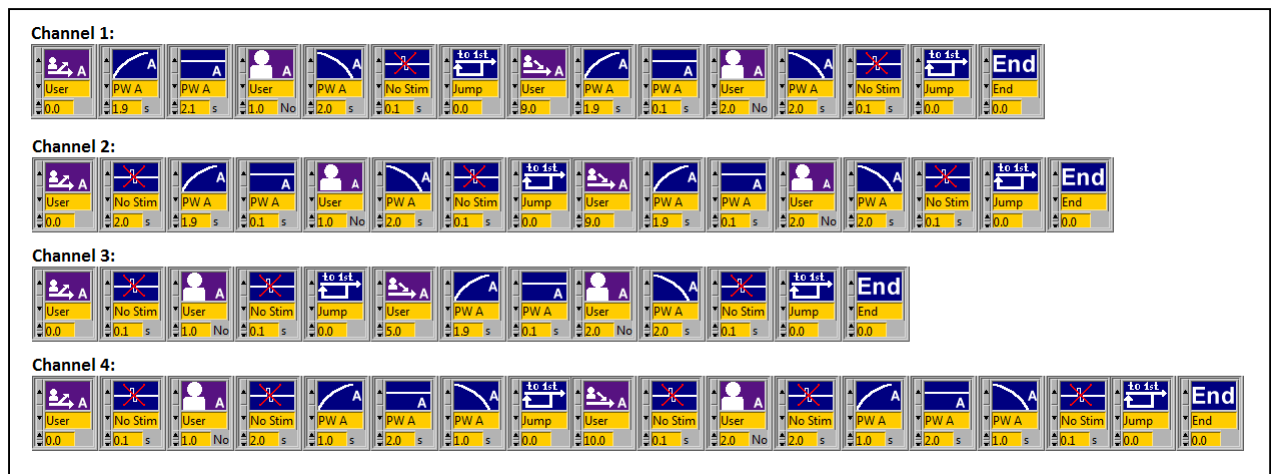
	User interaction (7)	Allows the user to control stimulation pattern execution through the use of external signals from sensors or man-machine interfaces. All channels are synchronized before user interaction is accepted.
	User branch (2 sets)	Allows the user to control between the execution of two different stimulation program sections using external signals from sensors or man-machine interfaces. All channels must be synchronized at this primitive (top) before a user branch can be executed. Bottom primitive shows where in timeline the optional branch will jump to if designated user control signal(s) are received.

**Table 9-1: Summary of supported stimulation primitives.**






This example describes a neuroprosthesis for grasping which had been developed for a 22-year old male individual with C5 complete spinal cord injury[36]. The patient was fitted with a neuroprosthesis that allowed him to perform both lateral and palmar grasps on demand. On the stimulator all four channels were used. Channel 1 was used to generate finger flexion by stimulating the *flexor digitorum superficialis* and the *flexor digitorum profundus* muscles. Channel 2 was used to generate thumb flexion by stimulating the *flexor pollicis longus* muscle. Channel 3 was used to produce thumb opposition by stimulating the *median nerve* and Channel 4 was used to perform hand opening by stimulating the *extensor communis digitorum* muscles.

The stimulation protocol for this neuroprosthesis, shown in Figure 9-2 below, could generate both lateral and palmar grasps on demand. The stimulation protocol started by waiting for a user command (  ), the user interaction had been configured through the use of a push button input. By pressing the button for less than 0.5 s, the patient issued the command to perform a lateral grasp and by holding the button for more than 1 s, the patient issued the command for palmar grasp. When the patient issued the command for lateral grasp, the stimulator continued and

executed the adjacent primitives within the timeline for each channel; thus stimulating finger and thumb flexors. At this point the protocol was waiting () for the user to issue the command (button press < 0.5 s) to release the current grasp. Once this command was received the stimulation was ceased and the protocol jumped () back to the start and waited for the next command.



**Figure 9-2: Hand grasping stimulation protocol**

If the patient issues the command for palmer grasp, the stimulator jumps to the branch primitive () in each stimulation channel and executes the primitives from this point on. Once again, the protocol waited for the user to issue the grasp release command, once received, the stimulator stopped stimulation and jumped back to the start. Other primitives used in this protocol include the ramp primitives (, ) which were primarily used to provide comfort to the patient by allowing for a smooth transition between no stimulation to stimulation and vice versa. Also the no stimulation primitives () were used to ensure that the specific channel had ceased stimulation at a specific point and as a buffer in order to timely synchronize the stimulation between multiple channels. Finally the constant PW primitive () was used to administer a

constant stimulation sequence in order to activate specific muscles to provide the necessary functions.

As illustrated in the example above, the stimulation protocol developer has the freedom to design protocols for various neuroprostheses with the use of a wide range of available functions/primitives. It is therefore evident that this flexibility required a streamlined encoding scheme to communicate the various protocols to the stimulator in an effective and efficient manner, primarily because of the limited computational and storage capacity within the stimulator unit. The next section will provide the details regarding the stimulation protocol encoding scheme that allowed for the required flexibility and efficiency in communicating stimulation protocols of various lengths and complexities in a streamlined method to the stimulator.

## **9.2 Stimulation Protocol Encoding & Packaging**

As described above, the HCP provider develops stimulation protocols using the Complex Motion GUI. Once the desired protocol was ready to be deployed, it had to be translated into a byte stream that could be interpreted and executed by the stimulator's controller subsystem. Upon transferring a stimulation protocol to the stimulator, the Complex Motion software encoded and packaged the protocol utilizing a pre-defined encoding scheme. The entire stimulation protocol was packaged into a byte sequence of 2 KB (2048 Bytes). The 2 KB encoded stimulation protocol size is a legacy component which originated from the Complex 2 stimulator. This stimulator utilized chip cards for storing stimulation programs, with a capacity of 2 KB each. The stimulation program was transferred onto one of these cards, which was then inserted into the stimulator for decoding and execution. Through the advancement of the encoding and

packaging scheme, the total size of the formatted protocol remained at 2KB even though the new generation controller did not utilize these chip cards and therefore did not require the size to be fixed at 2KB. The removal of this restriction will allow for future variations in the encoding scheme and the addition of supported functionality by the stimulation protocol.

The 2KB encoded stimulation protocol was divided into 8 - 256 byte blocks in which the last two bytes of each block ended with an additional checksum (further information regarding checksum processing see Section 6.4.1). The 256 byte size per block decision was primarily based on dividing the encoded protocol into groups of related information. Table 9-2 below provides an overview and summary of the data that was contained in each block. A complete block by block, byte-level breakdown can be found in Table A4-1 of Appendix 4: Stimulation Protocol Encoding.

Block #	Data and function related to primitives
1	Contains header information such as stimulator identification, subject specific information and data related to the following primitives: <ul style="list-style-type: none"> <li>• User interaction primitives</li> <li>• User branch primitives</li> <li>• Amplitude change primitives</li> </ul>
2	Contains default pulse amplitudes, maximum amplitudes, maximum PWs, default stimulation frequency and data related the following primitives: <ul style="list-style-type: none"> <li>• Constant PW primitives</li> <li>• Text primitives</li> <li>• Amplitude change primitives</li> <li>• Frequency change primitives</li> <li>• Amplitude, PW and frequency randomization primitives</li> </ul>
3	Contains look up table data for channel-specific amplitude control.
4	Contains ramp values related to ramp up and down primitives.
5	Contains string of primitives for stimulation channel 1.
6	Contains string of primitives for stimulation channel 2.
7	Contains string of primitives for stimulation channel 3.
8	Contains string of primitives for stimulation channel 4.

**Table 9-2: Block specific data in encoded stimulation protocol**

The first block held stimulator specific data which included identification codes, stimulator ID, and manufactured date and stimulation specific data including subject specific information, user interaction data, user branch data, and amplitude change time data. The second block contained most of the constant parameter data values used by the stimulation such as amplitudes, PWs, frequency and textual display information. The third block held amplitude look-up tables that were used to select stimulation amplitude values based on an input control signal which was usually managed by the patient. The fourth block contained channel specific PW ramp values; this information was used to provide gradually increasing and decreasing stimulation intensities by altering the PW of the stimulation waveforms.

Blocks five to eight represented the primitives for each stimulation channel, where block five contained primitives for stimulation Channel 1, block six for Channel 2, block seven for Channel 3 and block eight for Channel 4. Each channel could hold up to 254 primitives which make part of the overall stimulation protocol. A single primitive was encoded with at least one byte of data where its value could range between 0 and 255. Table 9-3 below provides a summary of the byte encoding ranges for each type of primitive. Some primitives required specific data located within the encoded protocol to be executed. For example, during the decoding and execution of a ramp primitive, the data in block # 3 was used to reconstruct the desired ramp. For complete details on the primitive encoding and parameter dependencies, refer to Table A4-2 in Appendix 4: Stimulation Protocol Encoding.

Primitives	Byte encoding	Primitives	Byte encoding
Ramps	0-127	Random amplitude	241-242
Constant PW	128-167	User branch	228-229
Loops	168-203, 212-215	User interaction	248-254
Delays	204-211	Push button	243-245
Amplitude change	216-219	Text	233-236
Frequency change	220-223	Sound	239-240
Random frequency	224-225	Turn off	230
Random PW	226-227	End Stim	255

**Table 9-3: Primitive byte encoding summary.**

The primitive encoding scheme is best illustrated and understood through the use of a simple example. The first step involved the development of a stimulation protocol, usually performed by a HCP, Figure 9-3 below is a screenshot of a sample stimulation protocol designed using the GUI software on a PC. This simple stimulation protocol was developed to illustrate the primitive encoding scheme which was carried out when a protocol was transferred to the stimulator. The

first stimulation channel involved the use of a one second ramp up (ramp A) then a constant PW (PW A) stimulation for five seconds, followed by a one second ramp down (ramp A). The second channel started with no stimulation for seven seconds followed by a ramp up, constant PW and ramp down which was identical to channel one (other than constant PW D) and a jump back to first primitive. Channel three used a two second ramp up (ramp B), then an eight second delay primitive, followed by a two second ramp down (ramp B) primitive. The fourth channel used a 1.6 second ramp up (ramp A), then a ten second constant PW (PW C) and a 1.6 second ramp down, this functionality was enclosed by a loop performed 5 times. This specific stimulation protocol would execute until the user stops the stimulation since it incorporated the “jump to first” primitive (infinite loop). In this case stimulation channels 1, 3 and 4 would finish their respective executions and channel 2 would continue until the stimulator was stopped.







**Figure 9-3: Example stimulation protocol to illustrate primitive encoding.**



Stimulation Channel	Primitive Byte Encoding (HEX)
Channel 1	09 81 84 85 29 FF
Channel 2	A1 A2 A6 09 99 9C 9D 29 ED FF
Channel 3	53 D0 D2 73 FF
Channel 4	A8 0F 92 95 96 2F A9 AB FF

**Table 9-4: Primitive byte encoding**

Once the protocol was finalized and ready for transmission to the stimulator, the primitives for each stimulation channel were encoded. Table 9-4 above contains the final outcome of the encoding for the protocol example given above. The byte encoding could be followed through the use of Table A4-1 and Table A4-2 in Appendix 4: Stimulation Protocol Encoding. The detailed encoding procedure for Channel 1 will be provided here. The first byte for this channel is “09”, which represents the “ramp A up” primitive (). This value also represents the number of cycles the ramp executes for, before reaching its final value (PW-A), where “0” represents one cycle, “1” represents two cycles, etc. Assuming the stimulator is configured in time based mode, each cycle will be measured in 100 ms intervals. Therefore the ramp would execute for ten cycles or  $10 \times 100 \text{ ms} = 1 \text{ s}$ . The next three bytes (81, 84 and 85 in hexadecimal (HEX)) represent the “constant PW A” primitive (). In base ten (decimal), these bytes translate to 129, 132 and 133 which correspond to keeping the constant PW A for  $200 \text{ ms} + 1600 \text{ ms} + 3200 \text{ ms}$  totalling 5000 ms or 5 s. The next byte is “29”, which represents the “ramp A down” primitive (). The byte translates to 41 in base ten, which corresponds to reaching a 0 PW in ten cycles or 1 s. Finally, the last byte “FF” represents the “end stimulation” primitive () for channel 1.

Once the encoded stimulation protocol was transferred to the stimulator, the controller decoded the various byte streams in order to re-compile the desired stimulation and execute the various

functions for each stimulation channel. The next section describes the design and implementation of the decoding and real-time execution procedures which were carried out on the LPC2368 microcontroller.

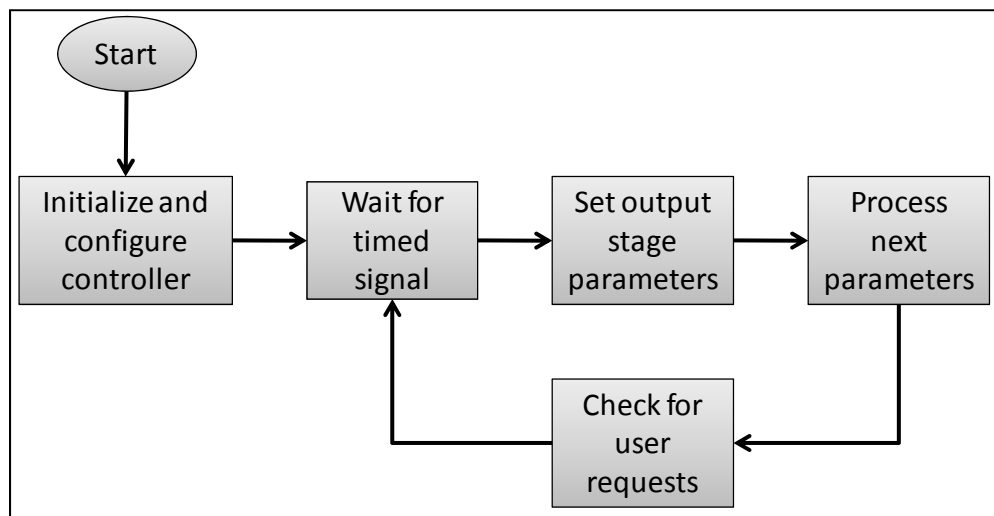
### **9.3 Stimulation Protocol Decoding and Execution**

The core functionality of the stimulator's controller was the timely and accurate decoding and execution of a stimulation protocol developed through the PC GUI software. This section will describe the details pertaining to the execution of a stimulation protocols, including the decoding and processing structure with real-time support. The discussion will start off by providing the necessary details on the overall methodology and structure, then analysis of specific function design and implementation will follow.

#### **9.3.1 Overall decoding and execution design structure**

When a stimulation protocol was loaded into the stimulator, the user can initiate the execution of the desired stimulation. During the execution cycle, the microcontroller consecutively decoded specific parts of the protocol as required until the stimulation program was completed or was stopped (or paused) by the user. When the user provided the start command, the microcontroller initialized and configured the decoding and execution environment, then waited for the real-time signal, upon receiving this signal, the initial stimulation parameters were provided to the output stage. The next parameters were then decoded and pre-processed, ready for the next timed signal. During this cycle, any pending user requests were also processed. The entire implementation of the decoding and execution processes is found in the “outputStageControl” and “stimulationProgram” classes.

Figure 9-4 below illustrates a flow chart of the protocol decoding and execution process that the microcontroller performs. This flow chart illustrates the normal decoding and execution procedure without any exceptions. Example exceptions include control input from the user to pause or stop the current stimulation. Other exceptions also include the user parameter change requests that provide changes in stimulation parameters during run-time, which are not within the stimulation protocol or parameter acquisition requests.



**Figure 9-4: High-level stimulation decoding and execution flow chart.**

At the start of any stimulation sequence, the “Initialize and configure controller” block was executed first. Within this block all data structures and variables used for decoding and execution were initialized and the maximum stimulation parameter values (amplitude and PW) were loaded from the protocol. Then all default initial stimulation parameters were set with a zero PW and exchanged with the output stage. Finally the real-time ISR used for the timed signals was initialized and the protocol decoding and execution process was started.

The “Wait for timed signal” block was implemented with the use of peripheral timer 1 (TMR1) on the LPC2368 microcontroller. TMR1 was integrated with an ISR to generate an interrupt at specific timed intervals. The ISR then triggered the stimulation processing task in order to indicate when to provide the next stimulation parameters to the output stage. On the LPC2368, the CPU clock was configured to 48 MHz (see Section 5.3 for clock frequency details). TMR1 operated at 12 MHz ( $\frac{1}{4}$  of the CPU clock) and was configured as a 32-bit incrementing counter. When the counter reached a specified value (timer match register –  $TMR_{CNT}$ ) an interrupt was generated. The  $TMR_{CNT}$  was calculated with the following equation:

$$TMR_{CNT} = \text{int}(TMR1_{CLK} / F_{OP}) - 1$$

Where “ $TMR1_{CLK}$ ” was the operating frequency of TMR1, in this case 12 MHz and “ $F_{OP}$ ” was the frequency that corresponds to the desired counter value. For example, in time based operation mode,  $F_{OP}$  would be initially set to 10 Hz (trigger output stage updates after every 0.1 seconds or 100 ms). In this case the resulting  $TMR_{CNT}$  was 1,199,999. The counter value started at zero, therefore the “-1” in the equation compensates for a single clock period. Once the counter value was reached, the interrupt was generated and the counter was reset back to zero and counting continued for the next timed signal event.

The next block entitled “Set output stage parameters” performed the updating of stimulation parameters for each channel at the output stage. Once the timer ISR provided the required timed signal, the stimulation parameters were transferred and synchronized to the output stage. Since the output stage was not currently ready for integration, these parameters were transmitted serially through the LPC2368’s UART1 interface. This data was then retrieved and analyzed on

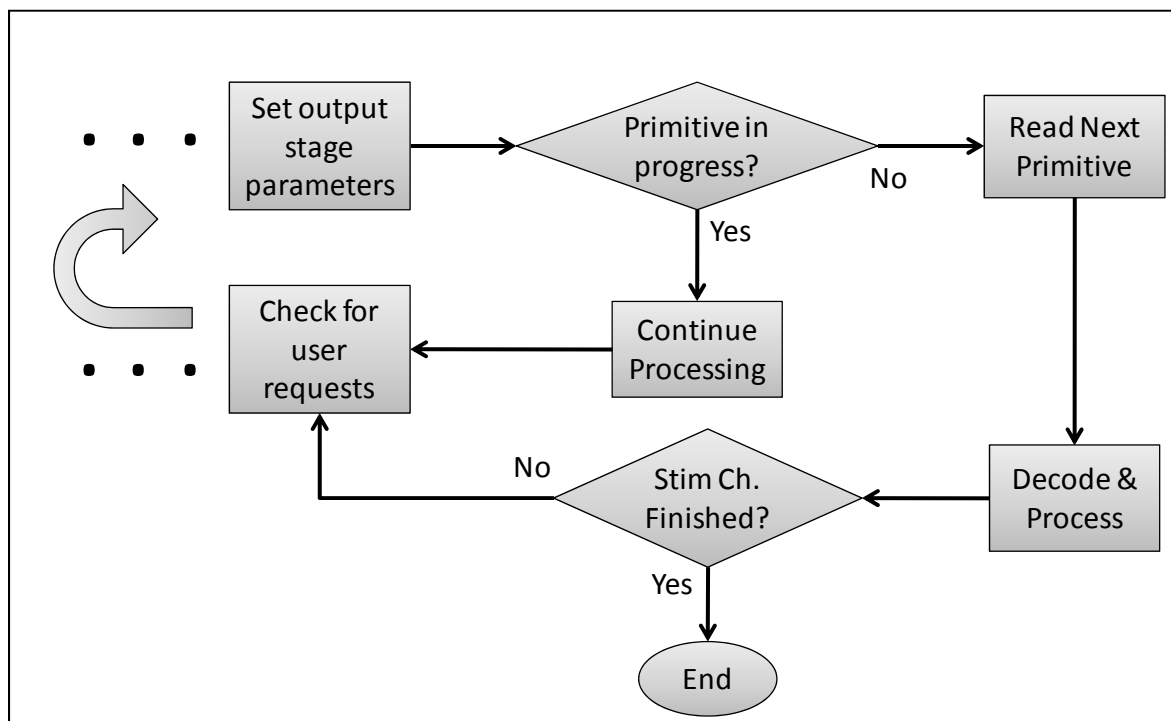
a PC with for example the use of terminal program such as Windows Hyper Terminal application. This information was used to verify and validate the stimulation parameters intended for the output stage. Information regarding the data retrieval through this interface for testing purposes can be found in Chapter 10.

Once the stimulation parameters had been synchronized with the output stage for the current timed sequence, the subsequent parameters were decoded, processed and calculated before the next triggered was received in the “Process next parameters” block. This was the main component in the stimulation protocol decoding and execution cycle within the microcontroller which decoded and executed the primitives associated with each stimulation channel. The processing resulted in updated stimulation parameter values and control characteristics of the stimulation protocol being administered. The subsequent sections will provide the design and implementation details regarding the decoding and execution related to each primitive currently supported by the stimulator’s controller subsystem.

The last component in the decoding and execution flow chart was the “Check for user requests” block. This block processed any pending user inputs received from either the input interface on the stimulator or commands received from the PC software. These requests included instructions to stop, pause, change or receive stimulation parameters. If a pause request was received, the PW for each channel was brought to zero instantaneously and the current state of the stimulation sequence was captured and saved. Once the resume command was provided, the saved state of the stimulator was re-loaded and the stimulator executed a one second PW ramp up function which started at 0 to the resumed value, primarily to avoid “shocking” the patient. At this point the decoding and execution cycle would continue.

### 9.3.2 Primitive decoding and execution

The primitive decoding and execution functions were performed in the “Process next parameters” block (from Figure 9-4 above). In this block, the primitives for each stimulation channel were parsed and processed based on the provided stimulation protocol. As described above, each primitive was encoded with at least one byte of data. The protocol therefore contained sequences of byte encoded primitives for each stimulation channel that when they were executed together result in the intended stimulation program. Figure 9-5 below provides a more detailed view of the “Process next parameters” block. This flow chart illustrates the logical sequences that took place when processing stimulation primitives. For simplicity, this diagram displays the steps performed for a single channel, although this method can be extended to any number of stimulation channels.



**Figure 9-5: Process next parameters block flow chart**

The sequence carried on from the high-level decoding and execution flow chart following the “Set output stage parameters” block to the “Check for user requests” block. Once a primitive was read from the protocol, it was parsed for identification. This identified a primitive’s specific function and purpose prior to further processing it. The primitive identification scheme used macro statements in the C programming language (`#define`). These macros checked the value of the current primitive against pre-defined primitive value ranges, and when executed, returned either TRUE or FALSE, depending on whether the current primitive was within its range or not respectively. For example, the delay primitive was identified through the use of the following macro:

```
#define PRIM_DELAY (currPrim >= DELAY && currPrim < (DELAY + DELAY_SIZE))
```

The “currPrim” held the byte value of the current primitive in question, “DELAY” and “DELAY + DELAY\_SIZE”, provided the range of byte values in which the delay primitive was identified, in this case 204 to 211 (in decimal encoding). When the PRIM\_DELAY macro was read, it would result in “True” if the current primitive analyzed was a delay primitive and false otherwise. When analyzing a specific primitive, only its pairing macro would result in “True”, all others resulted in “False”. Therefore once identified, it was processed for its intended function. The following sections provide details on the decoding and processing of specific stimulation primitives once they have been identified. All testing related to the functionality and operation of supported primitives are found in Section 10.3.

### **9.3.2.1 Turn Off and End Stimulation Primitives**

The turn off primitive terminated the current stimulation sequences and shuts down (powers off) the stimulator unit, therefore affecting all stimulation channels. In this implementation of the stimulator controller subsystem, the microcontroller and its interfacing circuitry were powered by the USB 2.0 connection (see Section 7.1) and therefore when the USB cable was connected, the unit was powered. When the turn off primitive was executed, the controller stopped the stimulation on all channels and returned the stimulator to idle state.

The end stimulation primitive was channel specific, therefore when executed it stopped the stimulation sequence on the specific channel by reducing the stimulation PW to zero within the current stimulation parameter update sequence. The rest of the channels would continue their execution while the decoding and processing procedure skipped the idle channel. The entire stimulation program ended once all channels had reached their respective end primitive or the turn off primitive was executed in any one of the stimulation channels. At this point the stimulator would re-enter its idle state, ready for further instruction from the PC/hand-held device interface or user interface directly on the stimulator unit.

### **9.3.2.2 No-Stimulation Primitives**

The no-stimulation primitive stopped any stimulation sequence on a specific channel for a pre-defined amount of time. When the HCP developed or customized a stimulation protocol, the no-stimulation primitive required one parameter when used, which represented the amount of time the channel would not stimulate for. The parameter was encoded within the primitive itself; the single byte value ranging between 160 and 167 (decimal) denoted a no-stimulation primitive.



The no stimulation time encodings are listed in Table 9-5 below. If the HCP entered a no-stimulation time which was not matched by one of the values in this table, multiple primitives denoting the no-stimulation function were used that sum up to the desired value. For example, if a no-stimulation time of 10 cycles was used, the encoding would result in bytes: 163, 161. The controller would then process a total no-stimulation time of  $8 + 2$  cycles resulting in the required 10 cycle delay.

Primitive byte value (Decimal)	No-stim (in cycles)
160	1
161	2
162	4
163	8
164	16
165	32
166	64
167	128

**Table 9-5: No-stimulation primitive encoding**

### 9.3.2.3 Constant Pulse Primitives

The constant pulse primitives were used to provide constant PW stimulation on the specific channel for a pre-defined amount of time. The HCP was able to choose between four programmable constant PW values which could be used for this primitive and were labelled as PW A, PW B, PW C and PW D. The time duration that the primitive was configured for would be decoded in the same manner as the no-stimulation primitives above (see Table 9-5).

Primitives with encoded byte values of 128-135 (decimal) denote constant PW A, 136-143 for constant PW B, 144-151 for constant PW C and 152-159 for constant PW D. The PW changes related to this primitive would be completed when the primitive was executed, therefore within

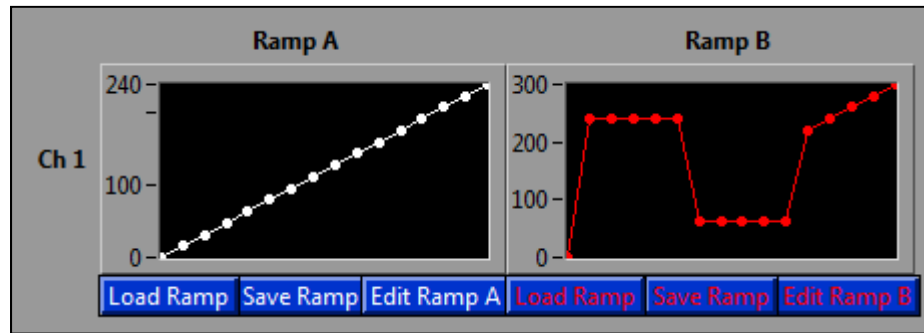
the immediate parameter update sequence. The programmed constants referenced by each constant PW primitive were encoded with two bytes of memory. Based on the required range of 0 to 1600  $\mu\text{s}$  with a resolution of 0.5  $\mu\text{s}$ , the two byte encoding provided a range of 0 to 32,768  $\mu\text{s}$  with 0.5  $\mu\text{s}$  resolution. The additional space left room for future extensions or increased resolutions without requiring more memory.

#### **9.3.2.4 Ramp Primitives**

The ramp primitives successively altered the PW of the stimulation signal on a specific channel following a programmable profile, resulting in linear or non-linear changes in PW. Its main application was to allow a smooth transition between one level of stimulation (no stimulation being a frequently used option) to another level of stimulation and vice versa. If ramps were not used in this case, the stimulation signal would be applied immediately which could cause patients to feel uncomfortable during these transitions. In order to provide a smooth start of stimulation, the ramp up primitives could be used which successively prolonged the PW over a time interval (e.g., 10  $\mu\text{s}$  to 300  $\mu\text{s}$  over 2 seconds). Similarly, the ramp down primitives could be used in the same fashion to successively shorten the PW at the end of a stimulation interval.

There were two programmable ramps (ramp A and ramp B) for each stimulation channel (4 channels with two ramps each, totalling 8 programmable ramps) that could be used for ramp up or down primitives. Figure 9-6 below displays both programmable ramps for Channel 1, as illustrated each ramp can be described with 16 discrete PW values. The 16 PW levels provided sufficient granularity to offer smooth stimulation transitions for patients, this had been proven successful with the application of the Complex Motion stimulator. When a specific ramp was

used in a stimulation channel, another required input variable was the time interval during which the ramp had to be executed, in other words to traverse the ramp from start to finish.



**Figure 9-6: Stimulation channel 1 programmable ramps.**

The ramp time variable was encoded in the primitive byte value itself (see Table 9-3 above), and the actual ramp profiles were located within the fourth block of the encoded stimulation protocol (see Table 9-2 above). Once the controller identified the current primitive as a ramp primitive, it isolated the specific ramp required and calculated the number of cycles that were required to traverse the ramp in the specified time (up to 32 cycles with 1 cycle step). Since only one ramp value could be executed per stimulation cycle (parameter update sequence) and depending on the time interval specified, some PW levels in the ramps had to be skipped or executed twice. Once the total number of cycles to execute the ramp was calculated, the spacing between each executed ramp value was then calculated. This calculation was performed by taking the total number of ramp values and dividing it by the number of cycles required to execute the ramp. For example if a ramp-up had to be executed in 7 cycles, the spacing between each executed ramp value would be  $16/7$  ( $\sim 2.29$ ). In the ramp up scenario, the last executed PW value had to be the last value in the ramp. For this example, the algorithm would execute the ramp values in the indexes listed in Table 9-6 from left to right.

2	5	7	9	11	14	16
---	---	---	---	----	----	----

**Table 9-6: Ramp up PW execution index values example.**

The equation used to calculate the values in Table 9-6 is:

$$Ramp_{up} = (\text{int}) \left[ \frac{T_{RV} \times C_c}{T_{RE}} \right]$$

Where “ $T_{RV}$ ” is the total number of ramp levels, “ $T_{RE}$ ” is the number of ramp values to execute for the specific primitive and “ $C_c$ ” is the current cycle. For the above example,  $T_{RV}$  is 16,  $T_{RE}$  is 7 and  $C_c$  started at 1 and increased by 1 until 7 ensuring all cycles were executed. When the ramp down scenario was used, the last executed PW value must be the first value in the ramp, therefore traversing the ramp in the opposite direction.

### 9.3.2.5 Amplitude Change Primitives

The amplitude change primitives altered the stimulation amplitude for a specific channel to a pre-programmed value when executed. Each stimulation channel had an initial default pulse amplitude used by the stimulation protocol. During a stimulation sequence, this amplitude could be altered by using an amplitude change primitive. There were four programmable amplitude values associated for each channel (amplitudes A-D). Depending on the required muscle response, a change in amplitude could generate stronger or weaker muscle contractions. The HCP could integrate the amplitude change primitives to alter the amount of muscle response during the stimulation. Another specification required when using these primitives was the transition time. Sudden amplitude changes like sudden PW changes could be uncomfortable or

painful for the patient, therefore including a transition time for the amplitude to change from its initial value to its final value made the difference with respect to comfort.

Constant Amps Ch 1 Transition			Constant Amps Ch 3 Transition		
Amplitude A	5	1.0 s	Amplitude A	10	0.8 s
Amplitude B	15	5.0 s	Amplitude B	15	2.0 s
Amplitude C	7	1.7 s	Amplitude C	20	2.5 s
Amplitude D	64	30.3 s	Amplitude D	25	13.0 s
Constant Amps Ch 2 Transition			Constant Amps Ch 4 Transition		
Amplitude A	10	2.0 s	Amplitude A	70	45.0 s
Amplitude B	25	1.6 s	Amplitude B	27	15.0 s
Amplitude C	4	0.5 s	Amplitude C	8	2.2 s
Amplitude D	85	127.9 s	Amplitude D	13	7.4 s

**Figure 9-7: Programmable amplitudes with transition times.**

As described above, each channel contained four programmable amplitude values with corresponding transition time. Figure 9-7 above contains examples of programmable amplitudes for the four stimulation channels. The supported amplitude range would depend on the characteristics of the output stage, the requirements list a range of 0-125 mA (1 mA step size) for this surface stimulation system. Therefore amplitude values were encoded using a single byte which provided the controller with a theoretical supported range of 0-255 mA with a 1 mA resolution. The transition time was encoded using two byte of data, which would provide up to 65,535 cycles (1 cycle step size). As an example, a 7.4 second transition in timed based mode (10 Hz) would be translated to a 74 cycle delay. The transition from the old amplitude to the new amplitude was carried out linearly.

The amplitude change primitives were encoded based on the specific amplitude value, for example ‘amplitude A’ had a corresponding primitive. Table 9-7 below contains the amplitude change primitive byte encodings. Since the primitive would reside on a specific channel, the controller algorithm isolated the channel-specific data referred to by this primitive. It calculated the required transition interval, acquired the final amplitude value and calculated the linear amplitude ramp based on the initial and final amplitude, and transition interval.

Primitive byte value	Amplitude change
216	Amplitude A
217	Amplitude B
218	Amplitude C
219	Amplitude D

**Table 9-7: Amplitude change primitive byte encoding.**

### 9.3.2.6 Frequency Change Primitives

The frequency change primitives changed the stimulation frequency within the stimulation sequence at the selected moment. Further to the default stimulation frequency, there were four additional stimulation frequencies that could be utilized through the use of these primitives. Since the stimulator used one stimulation frequency at a time, this primitive affected all stimulation channels regardless which timeline it resided in. For some applications it would be necessary to alter the stimulation frequency at a specific instance, for example a higher frequency was used when stimulating a sensory nerve to elicit a reflex, then a lower frequency was set to provoke muscle contractions through the stimulation of a motor nerve.

When using the frequency change primitives, there were no additional parameters required. The frequency change was set immediately when executed within the specific execution interval.

The supported frequency range was between 0 and 200 Hz (with a 1 Hz step size). The

frequencies were encoded using 2 bytes of data each, which provided sufficient granularity for future capabilities and applications. The frequency change byte encoding is listed in Table 9-8 below. Since these primitives were global, the byte encoding indicated the direct action to the controller.

Primitive byte value	Frequency change
220	Frequency A
221	Frequency B
222	Frequency C
223	Frequency D

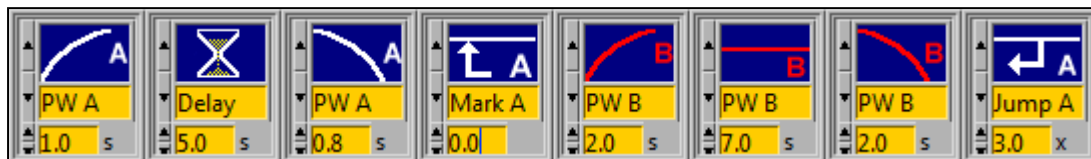
**Table 9-8: Frequency change primitive byte encoding.**

### 9.3.2.7 Delay Primitive

The delay primitive kept the current stimulation parameters from the previous function on the specified channel constant for a pre-defined period of time. For example, if the previous function was a constant PW, the same amplitude and PW were kept for the specified duration. If the previous function was no-stimulation, it continued with no stimulation. The main advantage of this primitive was when used after a ramp primitive. It took over the final PW of the ramp automatically without further specifications. Since the constant PW primitive was not used, it freed up these primitives for more specialized PW values when required. The delay primitive was encoded in a similar fashion as the no stimulation primitives (see Table 9-5 above) with primitive encodings values of 204 -211.

### 9.3.2.8 Jump Primitives

Many applications required that certain sets of primitives or all primitives in a stimulation channel needed to be repeated a specific or infinite number of times. This functionality was made possible with the use of jump primitives, without having to repeat functionality with duplicating sections of primitives. The first primitive in this group was the ‘jump to first’ primitive, which restarted the stimulation sequence on the selected channel by jumping back to the first primitive in the time line. Some applications required unlimited repetitions of channel specific stimulation tasks, this functionality could be programmed by the use of the primitive ‘jump to first’. When a specific section within a stimulation channel needed to be repeated a certain number of times, the ‘mark’ and ‘jump to mark’ primitives were used. The ‘mark’ primitive indicated the start of a loop region, and the ‘jump to mark’ primitive moved the execution sequence back to the corresponding ‘mark’ primitive. An example utilizing a set of ‘mark’ and ‘jump to mark’ primitives is shown in Figure 9-8 below. In this example the ‘PW B’ set of primitives were repeated three times with a total execution of four times (since the jump back does not take into account the first execution pass through).



**Figure 9-8: Jump primitive loop example.**

The byte encoding for the ‘jump to first’ primitive was 237 (decimal), once this primitive was decoded, the execution index was reset back to zero meaning the start of the current stimulation channel where this primitive was executed. The processing immediately executed the first



primitive in this timeline within the same parameter update sequence. The ‘mark’ and ‘jump to mark’ primitives were channel specific and there were four programmable sets (A-D) per channel. They could be cascaded to build single or nested loops. The encoding method used for each set of loops was the same, therefore the ‘marker A’ and ‘jump A’ set was used for illustration. The primitive encoding for the ‘A loop’ set is described in Table 9-9 below. The ‘marker A’ primitive was used as a placeholder indicating the start of the loop region, and the ‘jump A’ primitives were encoded based on the number of loops required. The total number of loops was encoded using the same method as the ‘no-stimulation’ primitive described above. The controller processed the total number of loops required for each type (A-D) and managed the execution sequence to follow the require execution paths.

Primitive encoding	Description
Marker A (168)	Index location
Jump A(169)	Loop once
Jump A(170)	Loop twice
Jump A(171)	Loop four times
Jump A(172)	Loop eight times
Jump A(173)	Loop 16 times
Jump A(174)	Loop 32 times
Jump A(175)	Loops 64 times
Jump A(176)	Loops 128 times
Jump A (212)	Loop infinite times

**Table 9-9: ‘Marker A’ & ‘Jump A’ primitive encoding**

### 9.3.2.9 Textual and Sound Primitives

During the execution of a stimulation protocol, there were instances where the patient had to be provided with some basic information or signal(s) to indicate an instruction or point of orientation. Such alerts could be provided through the use of visual and/or auditory means. The

controller was capable to provide textual information as well as auditory signals. These two modes of communications were carried through the text and sound primitives respectively.

During the administration of a stimulation protocol, it was often required to relay a brief message to the patient or therapist, such as welcoming the patient or instructing what to do next. For example, a simple message such as “push button to continue” could be issued at the beginning of the program or at a specific checkpoint. The current protocol included up to four (A-D) two lines, 16 characters per line textual messages. The HCP pre-configured these messages and inserted them at specific instances within the stimulation sequences on desired channels. Since there was one LCD display for the entire stimulator, all textual messages, regardless of which channel they originate from would be displayed in the same manner. When using the text primitives, a required parameter was the amount of time the message was displayed for.

Primitive encoding	Description
233	Display Text A
234	Display Text B
235	Display Text C
236	Display Text D

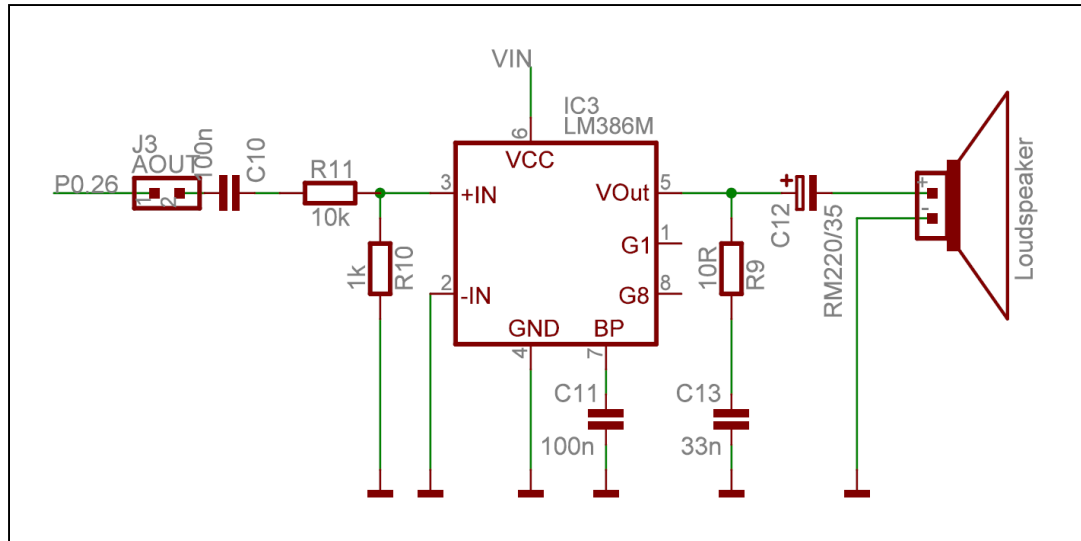
**Table 9-10: Text primitive encoding**

The text primitives were encoded in the protocol by the byte values (decimal) displayed in Table 9-10 above. Each text primitive also corresponded to its own display time, for example text A had a programmable display time that was pre-configured by the HCP and was the same regardless of which channel it resided in. The display time was encoded using one byte of data with a range of 1-256 seconds (1 second step size), providing sufficient flexibility when displaying a variety of messages to the patient. Once the text primitive was executed, the LCD

display was updated with the desired message and then, once the configured time had elapsed the preceding message was displayed again.

The sound primitives provided the patient and or therapist intermediate orientation of where the program was in the protocol's timeline. An example of such application could be to have a sound played when the stimulator was waiting for the patient to initiate a specific action or input. The sound provided the patient with an easy signal without having to consistently watch the LCD display for an instruction. The protocol supported up to two distinct sounds that could be integrated anywhere within the timeline of a specific stimulation channel and any number of times. These primitives would be executed in the same manner regardless of which channel they resided in, therefore they were not channel specific. The sound primitive also did not require addition parameters to operate, simply integrate within a timeline for operation.

The implementation on the controller included the use of the on board DAC interface and peripheral timer 0 (TMR0) on the LPC2368 microcontroller. The complete circuit interface diagram from the MCB 2300 evaluation board is shown in Figure 9-9 below. The output from the DAC (pin 6 on LPC2368) was interfaced to a low voltage audio power amplifier (LM386M). This power amplifier was intended for low voltage consumer applications and had its internal gain set to 20, keeping low external part counts. The current configuration was configured for a fixed gain of 20 (26 dB) for demonstration purposes, for final implementation a 10  $\mu$ F capacitor and variable resistor could be placed in series between pin 1 and 8 for variable gain control (volume knob).









**Figure 9-9: DAC to speaker interface diagram on MCB 2300 evaluation board (adopted from [60])**

The sound data used generated a series of four short sounds of increasing frequency, the data itself was arranged in an array of 16-bit signed values configured to be played at a frequency of 8 KHz. The peripheral timer TMR0 was configured to count and trigger an interrupt once every 8 KHz period. The ISR servicing TMR0 then loaded the appropriate data value from the sound data array to the output of the DAC, which generated the desired audible sound. Once the audio data had been executed, the TMR0 interrupts and DAC were disabled. For demonstration purposes, the two different sounds (sound A and B) were generated by executing the same sound data in reverse orders. Sound A was generated by traversing through the sound data from the lowest index to the highest and sound B by traversing it from the highest to lowest index.

### 9.3.2.10 Randomization Primitives

In some applications, especially those that stimulate a set of muscles for a prolonged period of time, slight variations in specific stimulation attributes could prevent muscle adaptation to the

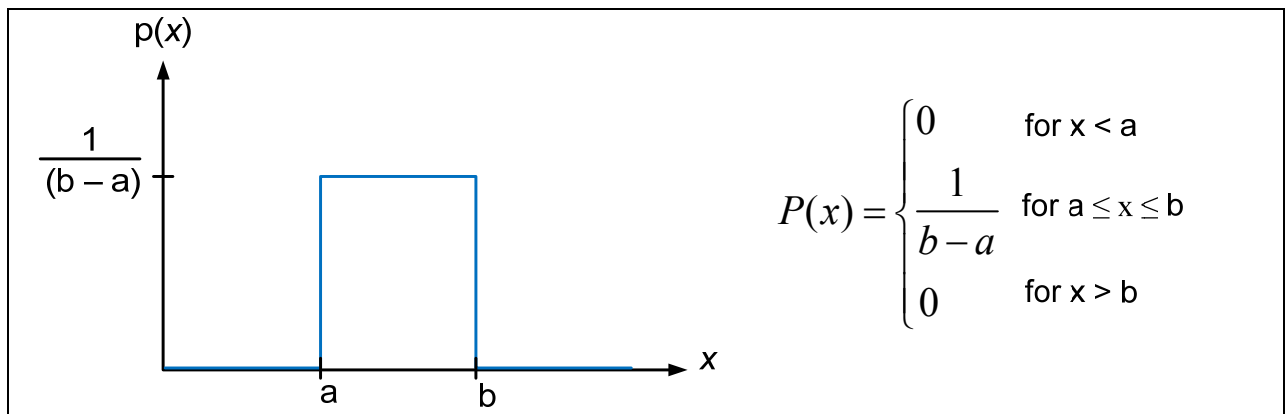
stimulus and fatigue. These stimulation attributes were directly related to the stimulus waveform administered to the patient and include the stimulation PW, amplitude and frequency. Each stimulation attribute had a corresponding randomization ON and OFF primitive and are shown in Table 9-11 below. The PW and amplitude randomization primitives were channel specific and the frequency randomization primitives affected all stimulation channels.

Attribute randomization	Enabled / Disabled	
PW		
Amplitude		
Frequency		

**Table 9-11: Stimulation attribute randomization primitives**

When the HCP incorporated an attribute randomization primitive, he/she had to specify the randomization amount in a percentage from the current value. For example, if the PW was 300  $\mu$ s and the desired random PW percentage was 10%, the PW would then vary randomly between 270  $\mu$ s and 330  $\mu$ s after every parameter update sequence. Following the randomization turn on primitive, the specified parameter would continue varying randomly until the corresponding randomization turn off primitive was executed. Each stimulation channel could be configured with one percentage variation value for PW and one for amplitude. Since the frequency randomization primitive was not channel specific, there was one configurable randomization percentage value included in the protocol. The allowable configurable percentage range was between 0 to 100% of the original parameter value.

The stimulation parameter randomization was carried out through the use of a pseudo-random number generator. For the purpose of the required functionality, the implementation of a complex hardware random number generator for true random number generation was not required. The generator used was based on the Lehmer random number generator algorithm found in the American National Standards Institute (ANSI) C library for multi-stream random number generation. This generator returned a pseudo-random number uniformly distributed between 0.0 and 1.0, its probability distribution function (PDF) is shown in Figure 9-10 below, with 'a' set to zero and 'b' set to 1. This software random number generation algorithm was chosen because it was able to provide a virtually infinite sequence of numbers that would satisfy almost any statistical test of randomness [64], therefore making it a valid option for this application. The period for this generator was  $(m - 1)$  where  $m = 2, 147, 483, 647$  (large prime number) and the smallest and largest values were  $(1 / m) \sim 0$  and  $1 - (1 / m) \sim 1$  respectively.



**Figure 9-10: PDF for uniform random number generator**

When a randomize attribute primitive was encountered, a seed value had to be provided to the random number generator in order to initialize its sequence of generated random numbers. If the same seed was utilized, the same sequence of random numbers would be regenerated. In order to provide some variation in the sequences of random numbers generated when multiple

randomization primitives were used, different seed values had to be chosen. Therefore, the seed value used when initializing the random number generator was the current count value of the TMR1 timer (see operation in Section 9.3.1) at the time of execution. Recall that this timer was used to control the timed sequence of the stimulation execution. The counter values ranged from 0 to 1,199,999 in timed based mode.

Once a randomize attribute primitive was executed, the specific stimulation attribute was set to change randomly between the desired range and the controller subsystem carried on to the next primitive within the immediate channel's timeline within the same processing interval. The decoding and execution of primitives continued with the randomization characteristics appended to the final stimulation parameter. The attribute randomization equation is provided below:

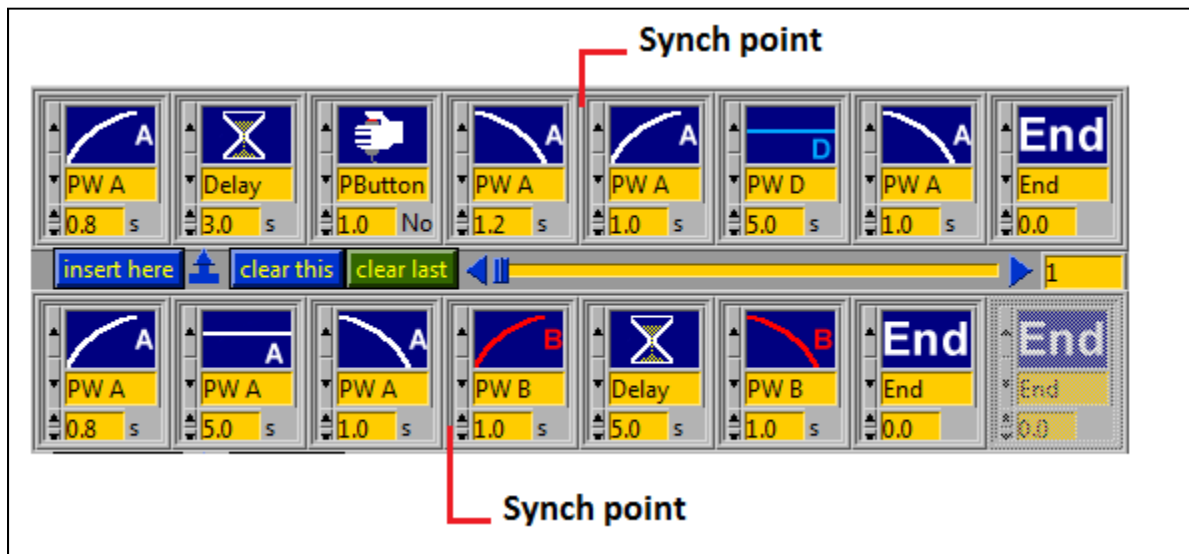
$$Att_{rnd} = Att_{cur} \left( 1 + (2 * rnd - 1) \times P_{\%} \right)$$

where “Att<sub>rnd</sub>” is the randomized parameter attribute, “Att<sub>cur</sub>” is the current parameter attribute, “rnd” is the result from the uniform random number generator that returns a value between 0.0 and 1.0 and “P<sub>%</sub>” is the required percentage randomization.

### 9.3.2.11 Synchronization Primitive

When different groups of muscles were stimulated in a specific temporal pattern, synchronization between stimulation channels was required. An intuitive method which is also commonly used to synchronize stimulation channels would be to ensure that the stimulation timing between all channels was carefully measured to the point where the synchronization was required, such that the desired temporal stimulation pattern would be provided. In certain

instances this method would not work, for example, if one channel was waiting for user input while the other channels were independently providing a set stimulation sequence. Since the timing of the required user input would not be known or guaranteed, the above channel synchronization technique would fail. Figure 9-11 below illustrates a common example scenario on a two channel stimulator (for simplicity). When the first channel was waiting for a push button input (discussed in the follow section) from the user, the second channel continued its execution independently. In this example, the HCP required the two channels to be synchronized at the points marked in this figure.

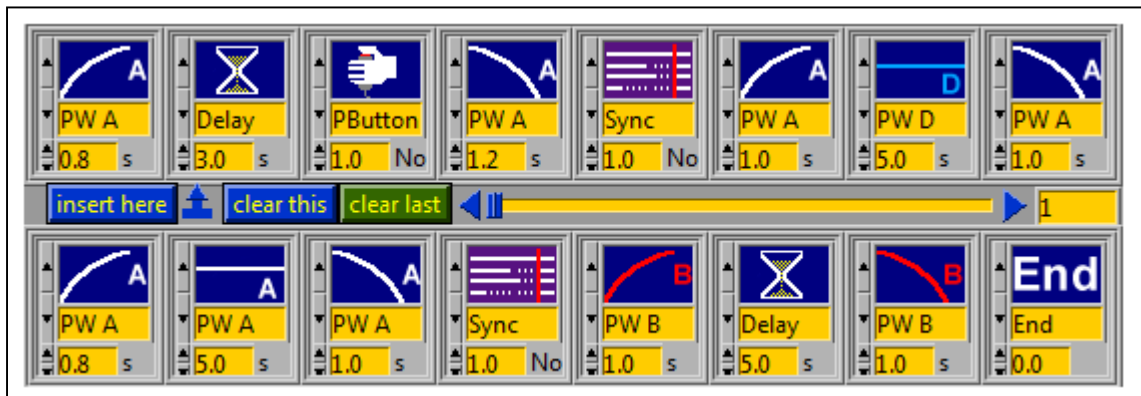


**Figure 9-11: Synchronization example on a two channel stimulator**

Without incorporating the synchronization primitives, the required stimulation timing between these two channels would not be guaranteed. Figure 9-12 below provides the solution by inserting the synchronization primitive at the desired “synch” point on each channel. The incorporation of this primitive would ensure that all stimulation channels had executed up to this point (synchronization primitive) before the stimulation sequences continued. The first channel to reach the synchronization primitive would stop executing additional primitives while keeping



the previous stimulation parameters constant (similar effect as the delay primitive). For example if the previous state was zero PW, the channel would be kept at zero PW until the rest of the channels reached their respective synchronization primitives before continuing.



**Figure 9-12: Synchronization example solution**

### 9.3.2.12 Push Button Primitives

A push button input is a user interaction which was often used as part of a neuroprosthesis as an input trigger to control the stimulation sequence. This functionality was provided by two push button primitives, specifically the channel specific push button (PButton primitive) and the synchronized push button (PB sync). The PButton primitive was used as a trigger for a single channel, the rest of the channels continue execution independently while the channel integrated with this primitive waited for the push button signal. Therefore, this primitive allows input control of a single channel. The PB sync primitive was used as a trigger for all stimulation channels, therefore all channels must reach their respective PB sync primitive before the input trigger was accepted. Once all channels had reached this primitive, they continue stimulating with the last parameters set by the previous instruction (PW, amplitude and frequency) while

waiting for the appropriate trigger. When the trigger was received, all channels would resume processing the next primitives.

The input generated from a push button primitive was considered a digital input since its trigger was detected from a voltage level change, the implementation and interface to the LPC2368 microcontroller can be found in Section 8.3. The current implementation provided one digital input, which was processed by the microcontroller, therefore all push button primitives were processed from this interface. In the case of the PButton primitives, where each channel could process an independent push button trigger, an input stacking method was used. The result provided a sequential input handling procedure, where the first PButton primitive processed would be linked to the first push button trigger event, the second PButton primitive was linked to the second trigger and so on. This solution provided the use of a single physical push button input to be utilized for various input triggers independently. For verification purposes four light emitting diodes (LEDs) were used to indicate if a specific channel was waiting for a push button trigger, the LEDs would light up in sequential order (based on which channel was waiting for a push button trigger). A fifth LED was used to indicate when all channels were synchronized and waiting for a push button trigger for the PB sync primitives.

A special case of the PB sync primitive was the TTL primitive. This primitive also synchronized all stimulation channels then waited for the appropriate falling edge signal provided by a push button input before continuing. The main difference was that once all channels reached this primitive, the current stimulation was stopped and the stimulation parameters for the next primitives were pre-processed and ready to be set immediately once the trigger was received. This special purpose input primitive was specifically designed for the pulse-based mode

applications (research based), where the next stimulation parameters would have to be set after each stimulation pulse period. Therefore the TTL method was used to save processing time and ensure the next parameters were ready at a trigger event. The TTL primitives were also used when the trigger signal originated from another hardware circuitry instead of a user directly.

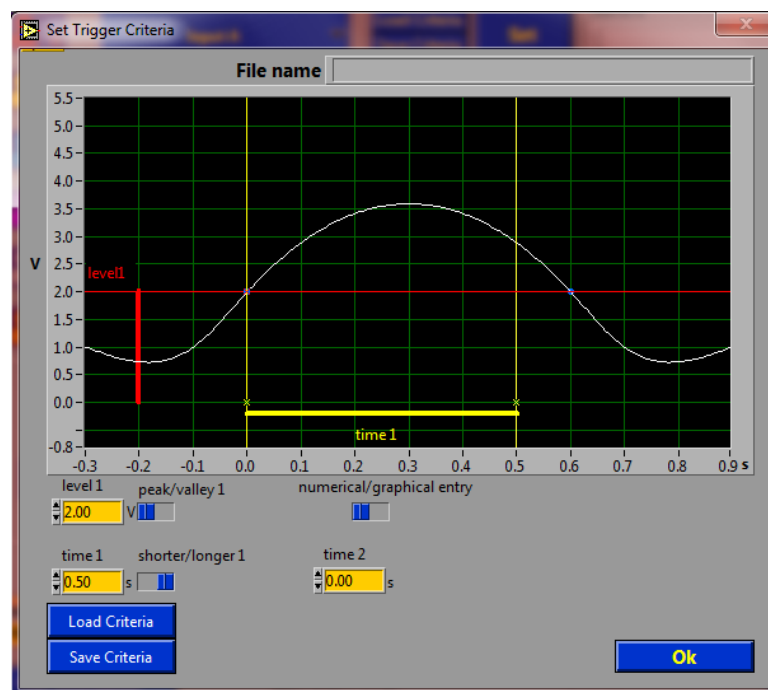
### **9.3.2.13 User Interaction Primitives**

In some applications more complex input control was required from external sensors or man-machine interfaces in order to capture a wider range of control signals. Such control signals could be generated from the subject, for example with the use of the Compex biofeedback sensor [25] (2M4456) voluntary muscle contractions could be captured as input to the stimulator. This input was then used to control the stimulation of the neuroprosthesis based on user commands. The user interaction primitives provided the link between identifying external control signals and the execution sequence of the stimulation protocol. Once a user interaction primitive was reached in a timeline, the execution of primitives was halted and the channel waited until the rest of the stimulation channels reached their corresponding user interaction primitive. When all channels had reached their corresponding user interaction primitive, the last stimulation parameters executed were kept constant (such as the PB sync primitive) while the stimulator waited for the specific trigger signal. Once the trigger criterion was fulfilled, all channels continued with the execution of the next primitives.

There were seven programmable user interaction primitives supported by the current stimulation protocol (A to G). All user interactions were interfaced through the ADC input interface on the LPC2368, a description for this interface is provided in Section 8.3. When using the user interaction primitives, trigger criteria must be programmed for each identifiable input signal. For example, one trigger signal could be generated by the subject pressing a button for a specific

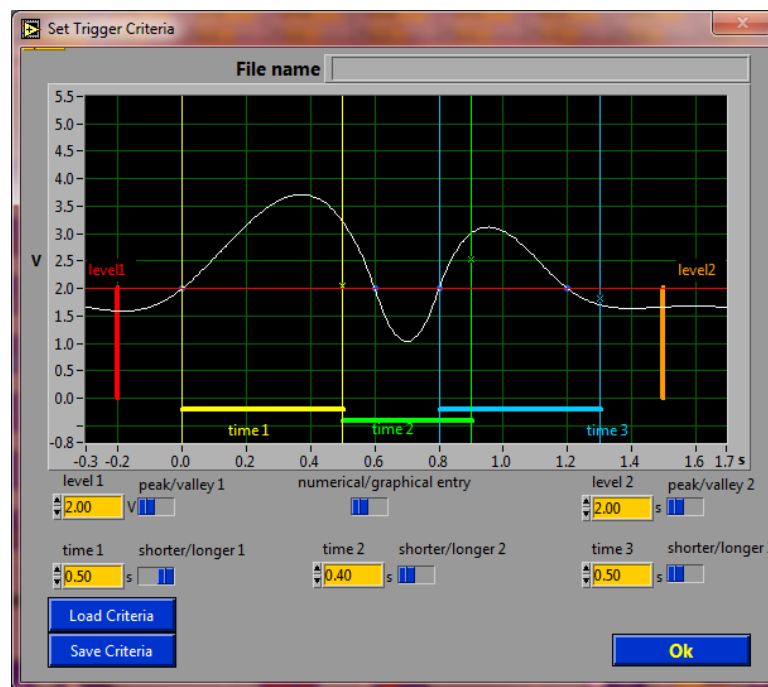
amount of time. The conditions that must be satisfied for this trigger to be successfully identified must be configured. The trigger criterion was configured through the use of the Compex Motion software on the PC.

There were four configurable parameters for simple trigger signal curves and ten parameters for more complex trigger signals. The simple trigger category included all input signals that were identified by a voltage level above or below a specific threshold for less then or more than a specified time. An example can be a push button that provided an input of 3 V, with trigger criteria of pressing the button for at least 0.5 seconds. In this case a threshold level of at least 2 V (since not pressing the button would provide an input level of 0 V) can be used with a trigger time set to at least 0.5 seconds. Figure 9-13 below illustrates the trigger criteria for this example, note that 'time 2' parameter was set to 0.00 s, this indicated that there were no further specifications required for this trigger.



**Figure 9-13: User interaction simple trigger example.**

The trigger criteria for more advanced input signal recognition uses ten parameters. In relation to the simple trigger, another change in signal identification is appended with a separation in time. Extending the push button example above for the more advanced signal recognition, once the button has been pressed for at least 0.5 seconds, after release, the second signal must start within 0.4 seconds. The second signal (pressing the push button) must fully arrive within 0.5 seconds. The specified trigger criteria can be seen in Figure 9-14 below. Note that since the 'time 2' parameter is not set to 0.00, the additional parameters for this trigger are shown and must be configured based on intended signal identification criteria.



**Figure 9-14: User interaction complex trigger example.**

The implementation in the stimulator uses the ADC converter to digitize the sensor data to logical voltage levels. This information was then used in conjunction with the parameters associated with the trigger specified by the primitives and the stimulation execution timer (TMR1) to re-compile the input waveform. When the compiled input waveform matched the

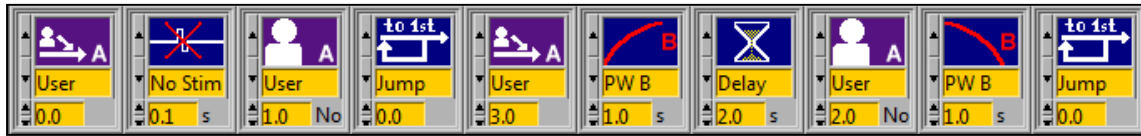
characteristics of the required input signal, the stimulator would trigger and all stimulation channels would continue to execute synchronously.

### **9.3.2.14 User Branch Primitives**

There were some circumstances where the use of a user interaction primitive to synchronize all stimulation channels and waiting for a specified trigger was not sufficient and a decision based on the input signal was required. Depending on the trigger signal received, this could be used to continue execution of primitives on each timeline or “jump” to another set of primitives further down the timelines that performed another action or function. This behaviour is realized with the use of user branch primitives, which in comparison to the user interaction primitives also synchronize all stimulation channels and waited for the trigger. The main difference was that two independent trigger signals or user interactions could be identified at this point and based on which was received this can decide on the stimulation sequences that follow.

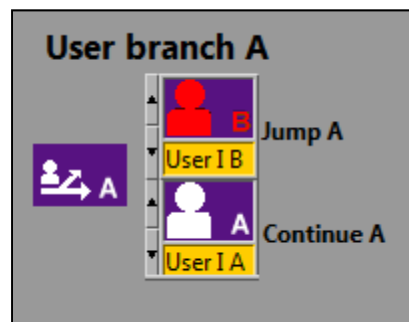
Some common applications for this functionality included walking or grasping neuroprostheses, in which different stimulation patterns were executed based on what was required. In the walking application, a specific stimulation sequence was required to perform the standing up action and another for gait support in walking. Patients with a grasping prosthesis often required the option to choose between lateral and palmar grasps. User branch primitives supported this functionality, where the choice between two program sections was provided based on external signals from sensors or man-machine interfaces. Figure 9-15 below shows a simple example from one stimulation channel where the user branch primitive was used to control which function was executed based on the input trigger. The first branch was no stimulation for the length of time controlled by another input signal (user interaction primitive), once received the stimulation

jumped back to the start. The second function was a stimulation sequence which kept the stimulation on (constant PW) until an input trigger stopped it and jumped back to the start.



**Figure 9-15: User branch primitive example.**

The current design supported two independent user branch primitives (A and B) where any of the seven user interaction primitive input trigger criteria could be used to control the continuation execution path or the jump path. An example configuration for the user branch A primitive is displayed in Figure 9-16 below. In this example, user interaction A indicated a continuation of execution for each channel, and user interaction B indicated a jump to the corresponding marker within each channel. The input signals on the stimulator were decoded in the same way as the user interaction primitives, except that two input signal patterns were analyzed simultaneously and whichever of the two was received first, the corresponding action would be carried out.



**Figure 9-16: User branch A trigger criteria**

## 10 Experimental Results

This chapter describes the experimental setup developed to gather results from the different component and system tests carried out to verify and validate the final design. The experimental results from these tests are then presented, analyzed and compared to the respective system requirements. The testing started with investigating the functionality and performance of the PC-stimulator interface. Then testing of the stimulation protocol decoding and execution was carried out, this part of the testing included the verification of the supported functionality including timing characteristics. The controller subsystem resulted in a highly-functional and flexible solution for use in FES systems, as an outcome there are a near infinite number of tests would be required to fully verify/validate all possible functionality in its entirety. The main objective of this chapter is to test the various scenarios that explore the individual components that make up a stimulation protocol in order to demonstrate the overall functionality and capabilities of the controller subsystem.

### 10.1 Experimental Setup

One of the primary objectives of effectively testing an embedded system was the ability to gather and extract useful data that confirmed the execution and results of the component under investigation, while minimizing interference with its operation. The experimental setup described here was the final test data acquisition implementation and was used for final component and system testing. During the implementation stage, further in-code debugging was used to acquire the necessary data in order to isolate problems. Implementation debugging is not covered, as it was naturally part of the development procedure. This experimental setup was



therefore utilized to gather data from the final system in order to illustrate how required functionality was met.

The experimental setup made use of four hardware components on the LPC2368 microcontroller three of which were used in the final implementation of the controller subsystem. The components used in the final implementation were the USB interface, the user output display component and peripheral timer 0 (TMR0). The component integrated exclusively for the experimental setup was the UART1 serial interface. Together these components made up the hardware base used on the LPC2368 microcontroller for collecting and acquiring data for testing. On the PC side, the experimental setup included the use of the legacy Compex Motion software to create custom stimulation sequences and download/upload them to the stimulator, the control center software to send instructions to the stimulator as well as receive immediate data on stimulation parameters and a Matlab program that received and processed the data through the UART serial connection from the microcontroller. Figure 10-1 below displays the overall experimental setup with the components used in the embedded system as well as the software used on the PC side.

In the embedded system, the USB 2.0 connection was used to interface to the Compex Motion software and Stimulator Control Center software on the PC, and was therefore used to validate this communication pathway. The LCD display, which provided user messages and system messages, was used to verify system states and associated text primitives. The UART1 provided a data link back to the PC where specific stimulation data was sent and processed. Finally, the TMR0 peripheral timer was used for tests when specifically looking at system timing

requirements and conditions. The timer was configured to operate at a 12 MHz frequency as it provided sufficient granularity and accuracy for the intended testing.

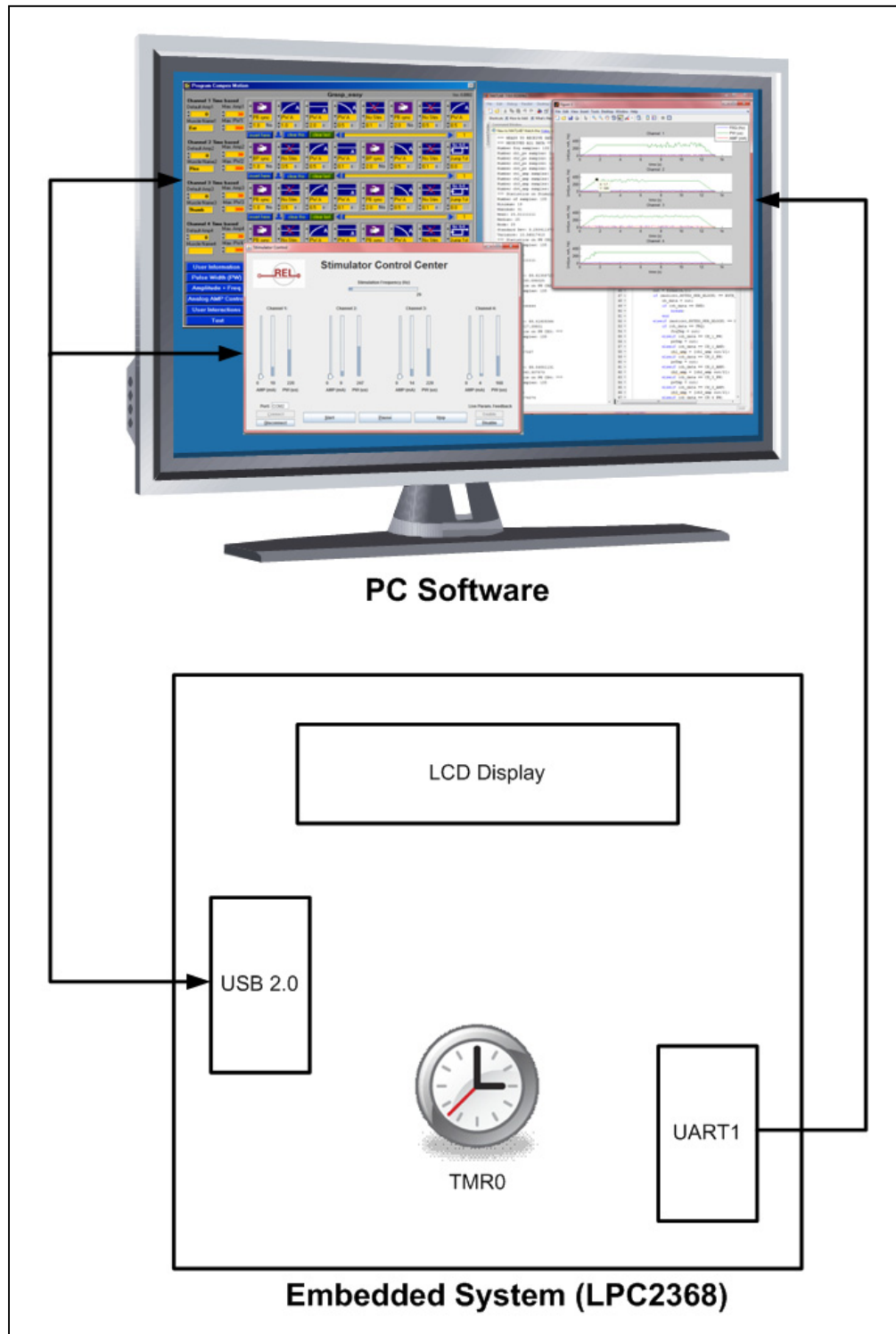


Figure 10-1: High level experimental setup block diagram.

The software used on the PC side included the Compex Motion software (see Section 7.2.1), the Stimulation Control Software (see Section 7.2.2) and the Matlab software program. Initially, the Matlab program was created to interface to the LPC2368 microcontroller in order to receive output stage data for debugging during development. During the development period this program had been extended to perform further processing on this data and generated visual representations of the stimulation of each channel. Since the output stage hardware component was not ready for integration, this interface to Matlab provided an effective representation of the output stage to ensure that the encodings delivered to Matlab through the UART interface were indeed accurate and represented the stimulation program that was being decoded and executed. It was understood that the UART interface provided only a “soft” communication pathway, as there were no hard real-time characteristics and timings that would be required when the final output stage was integrated and therefore this communication interface was solely utilized to interpret the final data directed to the output stage. Timing verifications had to be provided using other means and is discussed later. In the mean time, the controller transferred all stimulation parameters to the Matlab program after each stimulation trigger (as if it were the output stage) provided by timer TMR1 (see Section 9.3.1 for more details). The Matlab program therefore used this information to generate a pseudo-timed representation of this data.

The Matlab software was designed to receive data from the UART serial connection on the LPC2368, the data received was encoded data that would be transmitted to the output stage once integrated. Therefore, this software analyzed and decoded incoming data byte by byte as it was received. The data which represented the parameters for each of the four stimulation channels was then collected in data buffers for further processing. Initially, the program was designed to visually display the stimulation parameters for each channel by graphing the results as the data

was received. This method easily overloaded the graphical processing on the PC, making it impractical for longer stimulation protocols. Therefore, the data was then buffered with minimal processing within Matlab until the stimulation protocol had finished its execution, then this data was processed further and graphed with statistical relations presented. The statistical processing displayed the number of data values received, the minimum, maximum, mean, median, mode, standard deviation and variance for each stimulation parameter and for each stimulation channel. This information was particularly useful when analyzing the parameter randomization characteristics of the output waveforms. The experimental setup provided a means to acquire the necessary information about the controller subsystem in order to verify and validate its operation. Together these components were used in the various component and system tests that follow.

## **10.2 PC Interface Testing**

The PC interface was connected through the USB 2.0 port on the LPC2368 microcontroller. This interface provided the main connection to the controller subsystem where the full functionality of the stimulator was accessed through. The design and implementation details including the decisions of using USB as the communication medium were provided in Chapter 6. The supported functionality through this interface included the ability to communicate with the legacy Complex Motion software to receive and transmit stimulation protocols, retrieve stimulation parameters, change stimulation parameters, change the stimulation protocol, and finally to start, resume, stop and pause the stimulation.

The testing strategies involved executing the operation of each supported function in the communication protocol between the PC and stimulator in 30 separate trials. These tests were carried out to provide accurate timing information for the expected processing times associated with each supported function. For each test trial, TMR0 peripheral timer on the LPC2368 was

used to acquire precise timing information on each communication request through this interface. The timer was applied to accurately measure the time taken to complete a specific communication request. The time measurement was started from the instance when the PC and stimulator established a connection on a specific functional request and stopped when the request was completed and the communication interface entered the idle state.

With the use of the TMR0 timer, accurate timings were acquired when compared to other options such as timing the request from the PC, since the communication requests were acknowledged and terminated from the stimulator's side. In terms of minimizing interference with the intended microcontroller's operation, the use of this timer was only introduced at the beginning and end of the measured operation. Since the peripheral timers on the LPC2368 operate independent of direct software support, there was no additional interference with the execution of the program on the logical processor other than enabling and disabling the timer. The omitted overhead when using this timer was the number of CPU cycles required to enable and disable the timer. Based on the intended measurement these variations in measured time can be considered negligible and did not affect the final results.

All communication functions between the PC and stimulator were executed in 30 separate trials. This provided a sufficient number of timing samples that could be used for statistical representation. In all trials, the stimulation protocol used is shown in Figure 10-2 above, all stimulation protocols (regardless of functionality or stimulation length) were encoded into a fixed size of 2 KB, therefore each transfer was consistent in terms of data exchanged. In each trial the number of clock cycles was captured at 12 MHz from the TRM0 timer. The elapsed time in seconds was calculated by dividing the number of cycles by 12 MHz. The summary results of the communication test trials are captured in Table 10-1 below. This summary is

compiled based on the data captured from each test trial and this information including figures can be found in Appendix 5: Experimental Results Timing Data.

Communication Function	Average execution time (s)	Error margin (+/- s)
Legacy protocol download	2.391	0.0302
Legacy protocol upload	2.482	0.0477
Get parameter	3.02E-04	1.763E-05
Set parameter	6.655E-04	8.586E-05
Change protocol	1.672E-02	1.213E-04
Start stimulation	4.17E-03	2.807E-05
Resume stimulation	4.68E-03	8.215E-05
Stop stimulation	2.13E-05	1.842E-06
Pause stimulation	2.51E-05	1.407E-05

**Table 10-1: Stimulator/PC communication test timing summary.**

The legacy protocol download and upload functionality took place between the Compex Motion software on the PC which was originally utilized for the previous generation Compex Motion stimulator system and the controller. The new controller was designed to be backward compatible in order to provide convenience of directly transferring protocols from the original software platform. The average time for transferring a stimulation protocol to the stimulator in the 30 trials was 2.391 seconds with an error margin of 0.03 seconds. When transferring stimulation protocols to and from the Compex Motion stimulator, the average transfer time was roughly 15 seconds. The new controller subsystem was able to complete the same task about six times faster when compared to the Compex Motion system. The improvements in transfer speed were primarily due to the communication standards used in this interface. The new controller used USB 2.0 communications to/from the PC, where the Compex Motion used serial communications over a RS-232 connection.

The testing of the get (7 Byte instruction) and set (10 Byte instruction) parameter functionality was carried out on the stimulation amplitude of channel 0 in all 30 trials for consistency. The timing results regarding reading the parameter ( $\sim 300\ \mu\text{s}$ ) was performed on average about twice as fast when compared to changing its value ( $\sim 660\ \mu\text{s}$ ). The change protocol functionality had the ability to alter any component of the loaded stimulation protocol. This feature was supported if the stimulation protocol was not being decoded and executed simultaneously. The 30 test trials consisted of changing the stimulation title from “DA\_TransferTest” to “TEST 09”. The change protocol request instruction altered the entire 24 byte stimulation title block where the first seven bytes consisted of the new Title and the remaining 17 bytes were filled with zeros. The average stimulation protocol alteration completion time was 16.7 ms with a  $121\ \mu\text{s}$  error margin. To verify this change, the stimulation protocol was then transferred back to the Complex Motion software on the PC and the change in title was verified.

The remaining four communication functions from the PC were directly associated with controlling the execution of the stimulation protocol. These functions included the starting, stopping, pausing and resuming of stimulation. The testing involved repeatedly issuing each command using the Stimulation Control Center software on the PC. The stop and pause requests had an average completion time of  $21.3\ \mu\text{s}$  with  $1.8\ \mu\text{s}$  error margin and  $25.1\ \mu\text{s}$  with  $14.1\ \mu\text{s}$  second error margin respectively. The completion time of executing a pause request was longer since the current stimulator state had to be saved, where in the stop request the stimulation parameters were set to zero and the controller returned to idle state immediately. The start and resume request had an average completion time of 4.17 ms with  $28\ \mu\text{s}$  error margin and 4.68 ms with  $82\ \mu\text{s}$  error margin respectively. The start and resume execution times were considerably



longer when compared to the stop and pause performance primarily due to state configuration and loading of appropriate values from the immediate stimulation protocol.

### **10.3 Stimulation Protocol Decoding and Execution Testing**

In this section the testing related to the decoding and execution of stimulation protocols is covered. The testing strategy involved the composition of various stimulation protocols which covered the full extent of the functionality provided by the controller subsystem. These protocols were loaded individually into the controller and executed. The stimulation output was then captured through the use of the Matlab software described in Section 10.1. This output was then analyzed and compared to the initial protocol for verification and validation of its execution. Once the decoding and execution functionality of the controller subsystem has been presented, further testing will follow focusing on the timing characteristics related to the protocol decoding and execution.

#### **10.3.1 Stimulation Protocol Function Testing**

The stimulation protocol function testing relates to the accurate decoding and execution of the various primitives supported by the system, a complete summary can be found in Table 9-1 of Section 9.1. The stimulation protocols created to test the functionality of the controller do not have any direct practical application in FES therapy and were exclusively utilized to demonstrate the capacity and functional potential of the controller subsystem. The principle reason for the composition of these protocol tests was to validate and verify the complex decoding and execution structure within the controller subsystem. This ensured that the computed results precisely matched the intended stimulation protocols. The collection of stimulation protocols

chosen to demonstrate this functionality were not intended to cover all possible combinations of all 56 supported primitives and respective parameters as it was not the intended purpose for this testing strategy.

Parameter	Value
Default Protocol Title	stimProtTest00
Stimulation Mode	Timed based (100 ms segments)
Def. Freq. (Hz)	25
Freq. A (Hz)	20
Freq. B (Hz)	30
Freq. C (Hz)	65
Freq. D (Hz)	80
Rand. Freq. (%)	0*
Text A (16 char/line)	Text A line 1...Text A line 2...
Text B (16 char/line)	Text B line 1...Text B line 2...
Text C (16 char/line)	Text C line 1...Text C line 2...
Text D (16 char/line)	Text D line 1...Text D line 2...
Text A (s)	1
Text B (s)	2
Text C (s)	3
Text D (s)	4

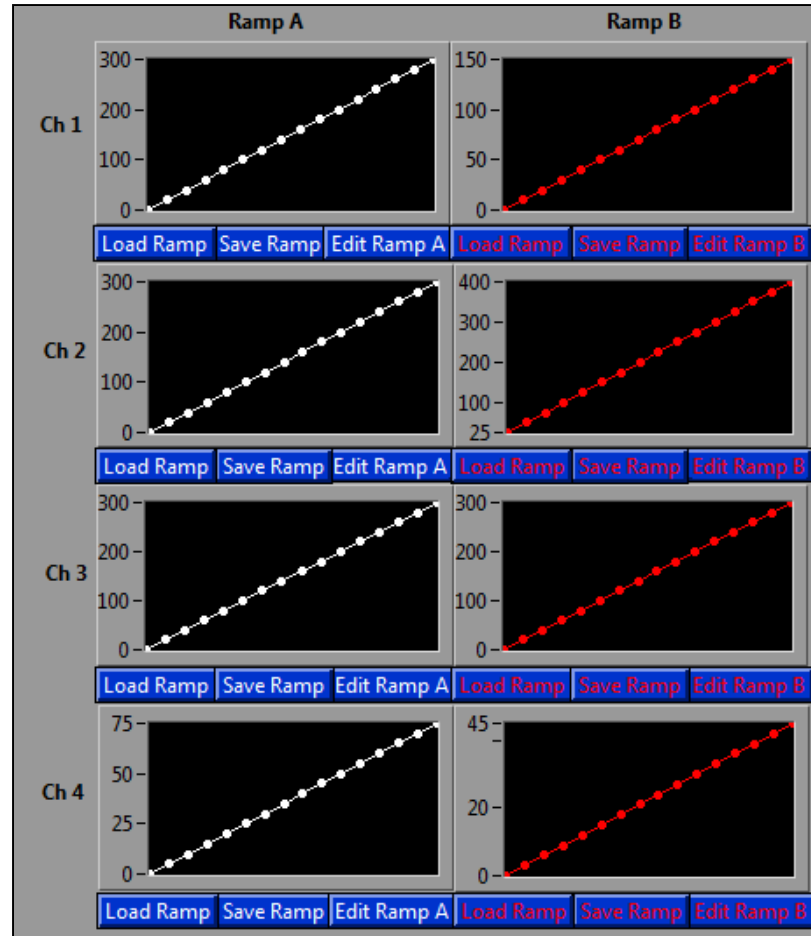
**Table 10-2: Global default stimulation parameters**

To start with, a default blank stimulation protocol which did not contain any primitives in its channels was created and used as a starting benchmark for all following stimulation protocols. This protocol was named “stimProtTest00”. This base benchmark provided all default parameters that were used in all consecutive stimulation testing. Table 10-2 above contains all global default parameters, these parameters apply to all stimulation channels and Table 10-3 below contains all channel-specific parameters. Note that all parameters with a ‘\*’ (randomization related parameters) were subject to being altered and appropriate changes would

be provided in specific tests when required. The custom PW ramp configurations for ramps A and B are shown in Figure 10-3, these were the default ramps used in the following tests.

Parameter	Channel 1	Channel 2	Channel 3	Channel 4
Def. Amp. (mA)	10	7	25	87
Max Amp. (mA)	30	15	45	120
Amp. A (mA)	5	5	10	45
Amp A trans (s)	0.5	0.5	1.0	3.0
Amp. B (mA)	15	15	15	70
Amp B trans (s)	1.5	2.0	1.5	5.0
Amp. C (mA)	20	8	30	80
Amp C trans (s)	1.7	0.8	2.0	7.0
Amp. D (mA)	25	10	40	100
Amp D trans (s)	2	1.0	2.5	10.0
Rand. Amp. (%)	0*	0*	0*	0*
Max PW ( $\mu$ s)	300	500	250	100
PW A ( $\mu$ s)	300	400	250	50
PW B ( $\mu$ s)	250	250	200	20
PW C ( $\mu$ s)	200	300	150	75
PW D ( $\mu$ s)	150	150	300	100
Rand. PW (%)	0*	0*	0*	0*

**Table 10-3: Channel specific default stimulation parameters**



**Figure 10-3: PW configurations for ramps A and B.**

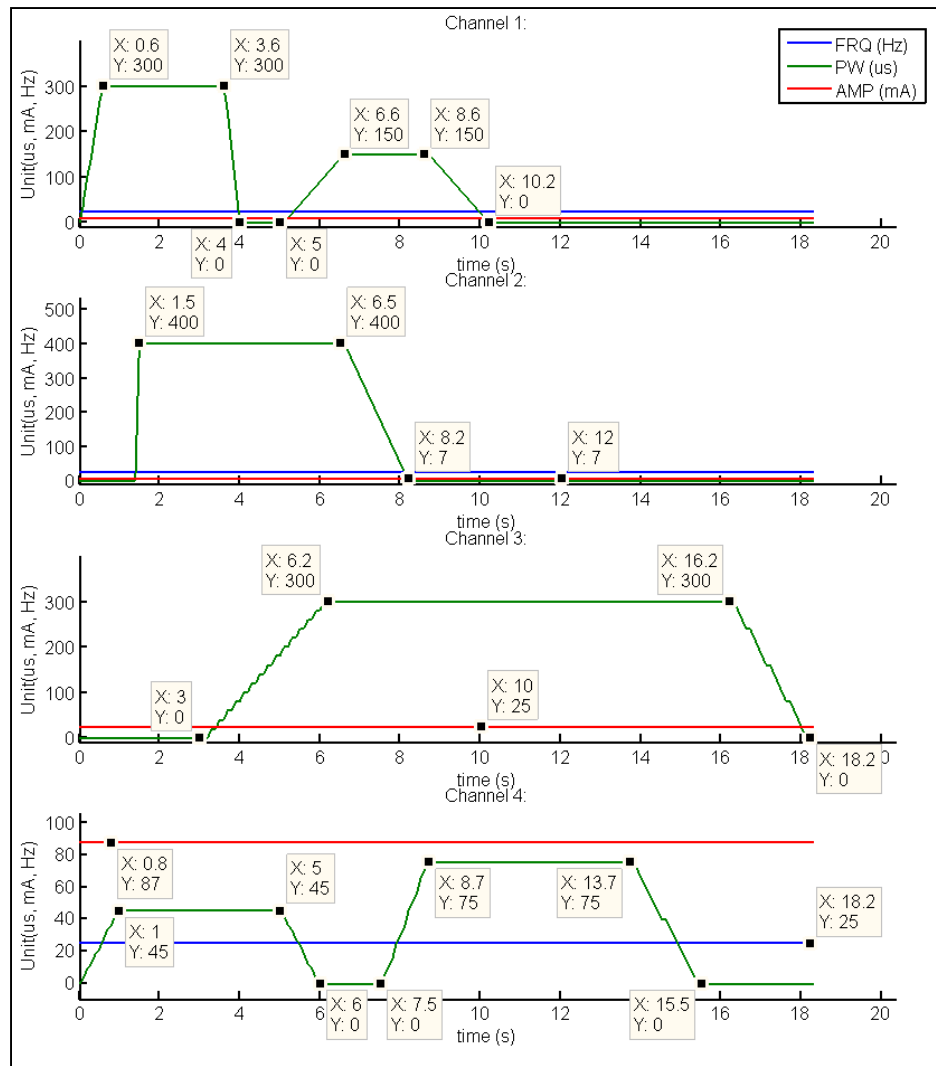
The stimulation protocols were created and downloaded to the controller subsystem using the Compex Motion PC software. Stimulation execution was controlled using the Stimulation Control Center software. The output data resulting from the execution of a stimulation protocol was collected through the UART1 RS-232 serial interface on the LPC2368 microcontroller with the Matlab analysis software. The Matlab software then generated a graphical representation of the output stage data which sped up the analysis process when comparing the final output information to the initial stimulation protocol. The LCD display interface to the controller provided visual information such as the stimulator's current state, status messages, error messages and text primitive related user information. Together these components from the

experimental setup were used to execute and retrieve information related to the various executed stimulation function tests.



**Figure 10-4: stimProtTest01 stimulation protocol.**

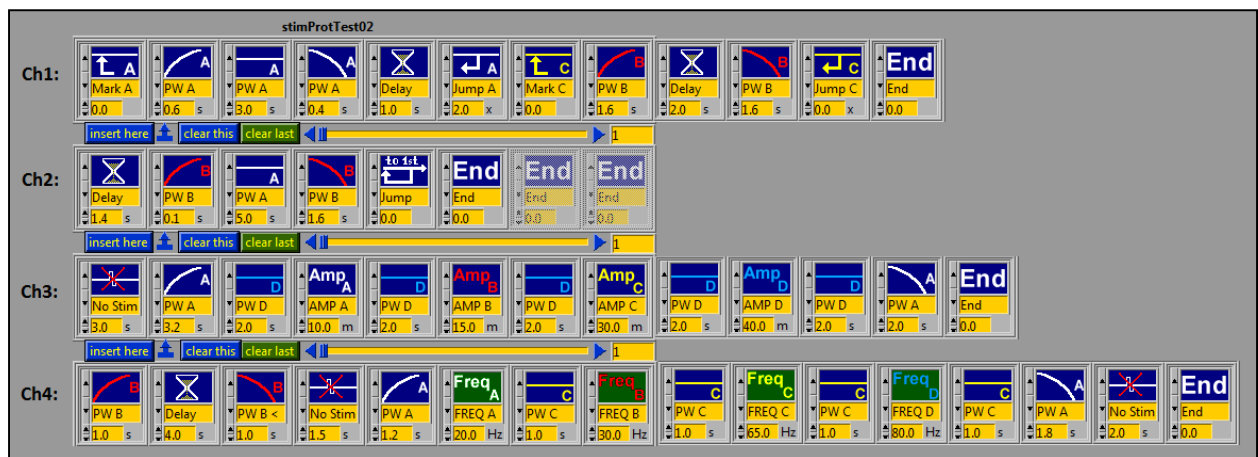
A total of five separate stimulation protocol tests were compiled and executed, the results from each were collected and compared to the original protocols for verification. The first test included the incorporation of the most common FES primitives such as ramps, constant PW and delays. Figure 10-4 above displays the stimulation protocol for this test. The functions performed by these primitives can be found in most stimulation protocols. The protocol execution results captured and processed by the Matlab program are shown in Figure 10-5 below.



**Figure 10-5: stimProtTest01 stimulation results captured from Matlab program.**

When comparing the output captured from Matlab and the initial protocol, the main characteristics to indicate whether the protocol was correctly decoded and executed are by analyzing the values for each stimulation parameter on each stimulation channel with respect to time. Recall that the Matlab time was generated based on the received data through the UART serial interface and therefore did not represent a real-time execution capture. When comparing the stimulation protocol to the graphs generated by Matlab, the timing characteristics and stimulation parameters match. For example, analyzing at the stimulation Channel-3 protocol and corresponding graph, the stimulation started by providing a PW of zero for 3 seconds (no-stim

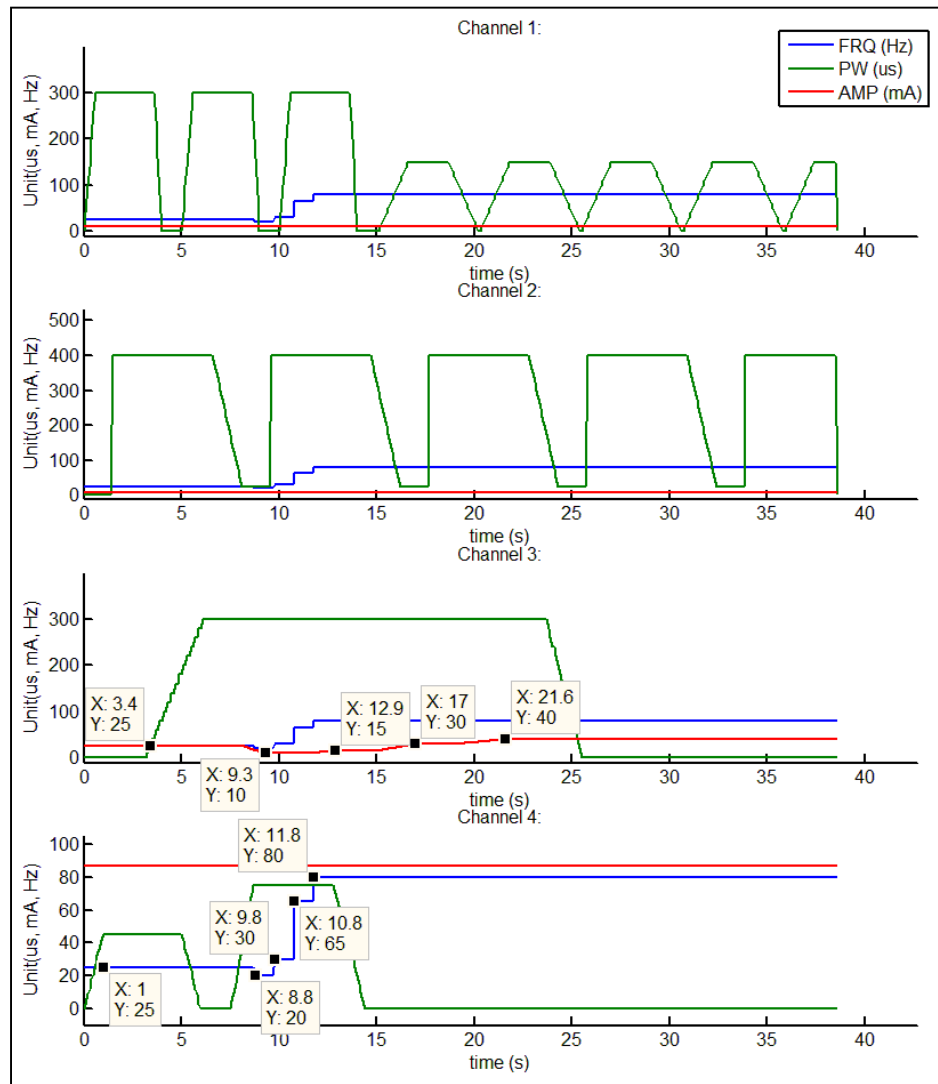
primitive). Then the ramp-up A primitive was executed over a period of 3.2 seconds until the final ramp value was reached at time 6.2 seconds. When looking closely at the ramp, it appears to be in a stair case configuration since each ramp contained 16 ramp values, therefore in 3.2 seconds at 100ms sequences all ramp values were kept constant for two 100ms intervals. Then the constant PW D (300  $\mu$ s from Table 10-3 above) primitive executed for 10 seconds followed by the ramp-down A primitive for two seconds.



**Figure 10-6: stimProtTest02 stimulation protocol.**

The second protocol test shown in Figure 10-6 above (named “stimProtTest02”) involved the use of loop, amplitude and frequency change primitives. Loops are most commonly required when specific functionality had to be repeated either for the entire stimulation sequence or a specific number of times. Amplitude and/or frequency changes were at times used when changes in intensity or function isolation were required. The output captured by Matlab is shown in Figure 10-7 below. In this protocol channel one looped the first four primitives three times and looped the next three primitives until the stimulation was ended by the user. Channel 2 shows the use of the jump to first primitive, which looped the prior set of primitives repeatedly, also until the user ended the stimulation. Channel 3 shows the use of the amplitude change primitives, each

amplitude change corresponded to a transition time (see Table 10-3). Channel 4 was used to change the stimulation frequency, note that the stimulation frequency was not channel specific and affected all channels when altered.



**Figure 10-7: stimProtTest02 stimulation results.**





**Figure 10-8: stimProtTest03 stimulation protocol.**

The third protocol test shown in Figure 10-8 above, named “stimProtTest03” made use of the global sound, text and randomization primitives. Since the sound and text primitives provided auditory and visual confirmations respectively, these were monitored with respect to the channel-specific stimulation sequences while running the test. The captured output stage stimulation results from Matlab are shown in Figure 10-9 below. The randomization parameters were configured as listed in Table 10-4 below. The parameter randomization functions provided fluctuations in the specified parameter, which for example, would be used for prolonged stimulation therapy such that the targeted muscle(s) do not adapt to the stimulus.

<b>Rand. Parameter</b>	<b>Percentage (%)</b>
Frequency	30
Ch1 PW	25
Ch2 PW	15
Ch3 PW	10
Ch4 PW	20
Ch1 Amp	25
Ch2 Amp	0
Ch3 Amp	25
Ch4 Amp	0

**Table 10-4: Randomization parameter summary**

Looking at the output waveforms in Figure 10-9, Channel 2 started with a PW randomization primitive at 15% from the ramp up primitive to the first delay primitive (6.6 s). During the first delay primitive, where the nominal PW was 400  $\mu$ s plus 15% randomization, the theoretical range was between 340  $\mu$ s and 460  $\mu$ s. Within the five second span, the lowest and highest PW were 342  $\mu$ s and 454  $\mu$ s respectively. Similarly, during the 30% frequency randomization, the range of random frequency values generated was between 18 and 32 Hz in the 11.6 second period with the default frequency being 25 Hz. Note that in Channel 1, the maximum allowable PW was 300  $\mu$ s, therefore during the randomization of the PW on this channel, the maximum value never exceeds this limit. The “Sound A” alert and “Text A” message (shown for 1 s) were observed after channel one completed its ramp up primitive (1.6 s) “Sound B” alert and “Text B” message (shown for 3 s) were observed once the PW randomization started in Channel 1.

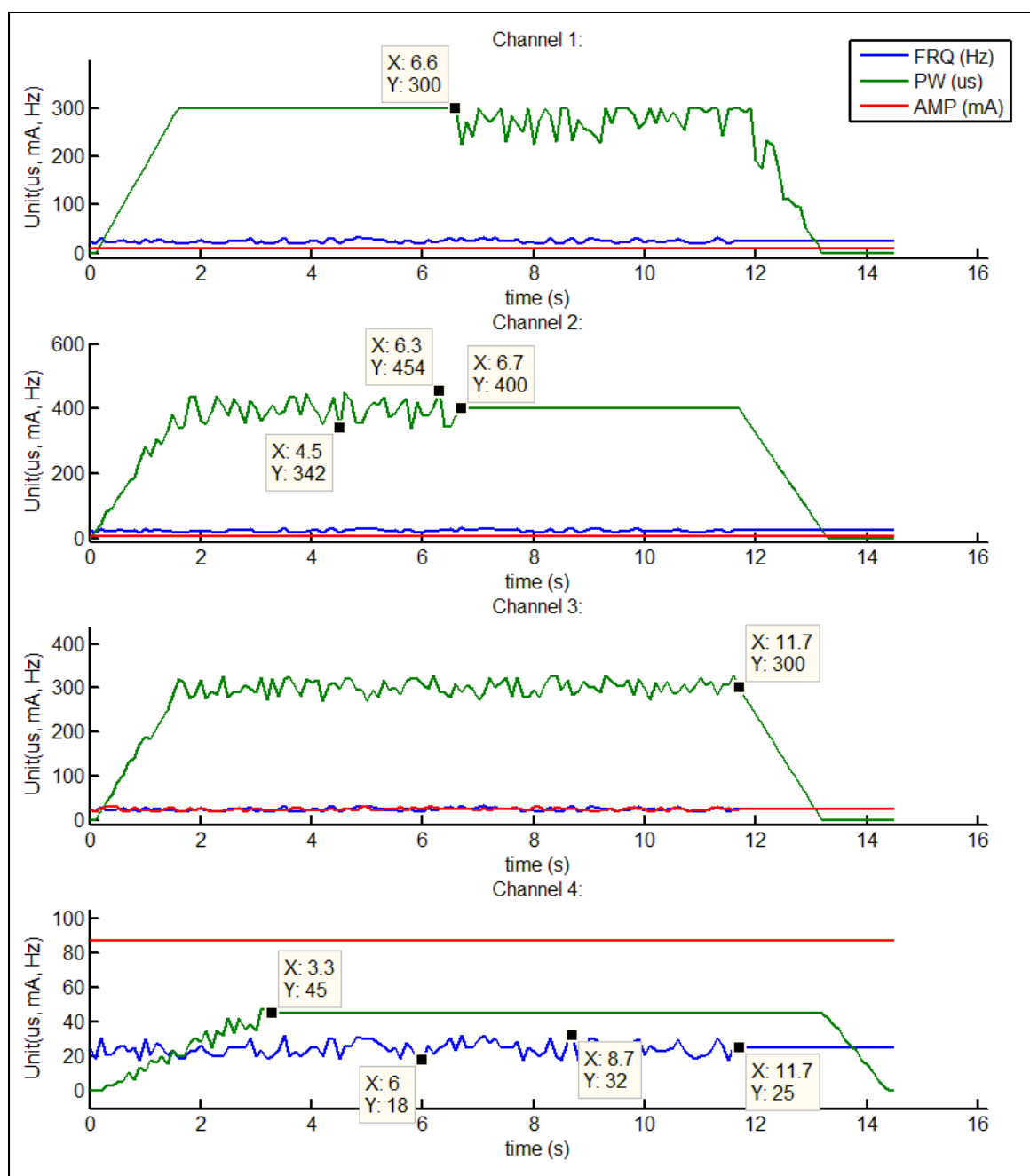
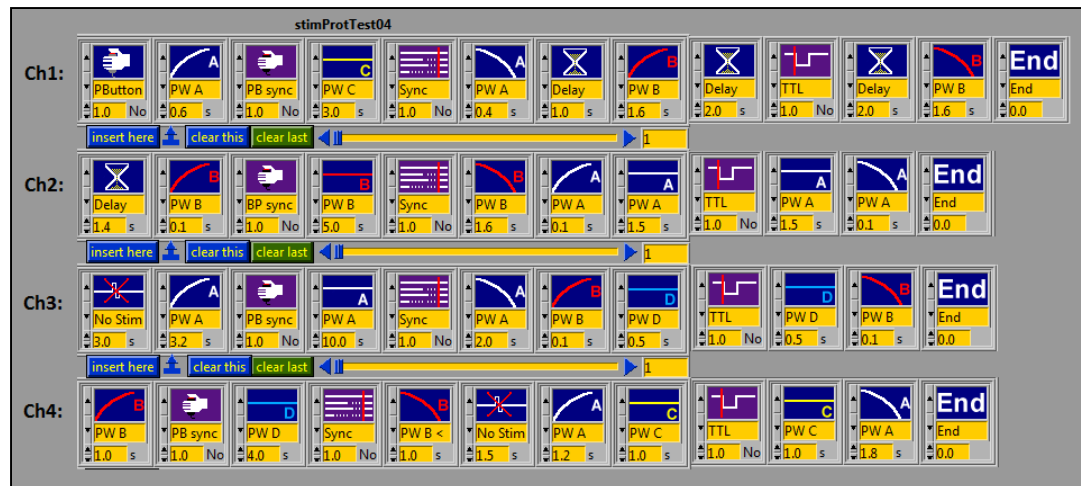


Figure 10-9: stimProtTest03 stimulation results.

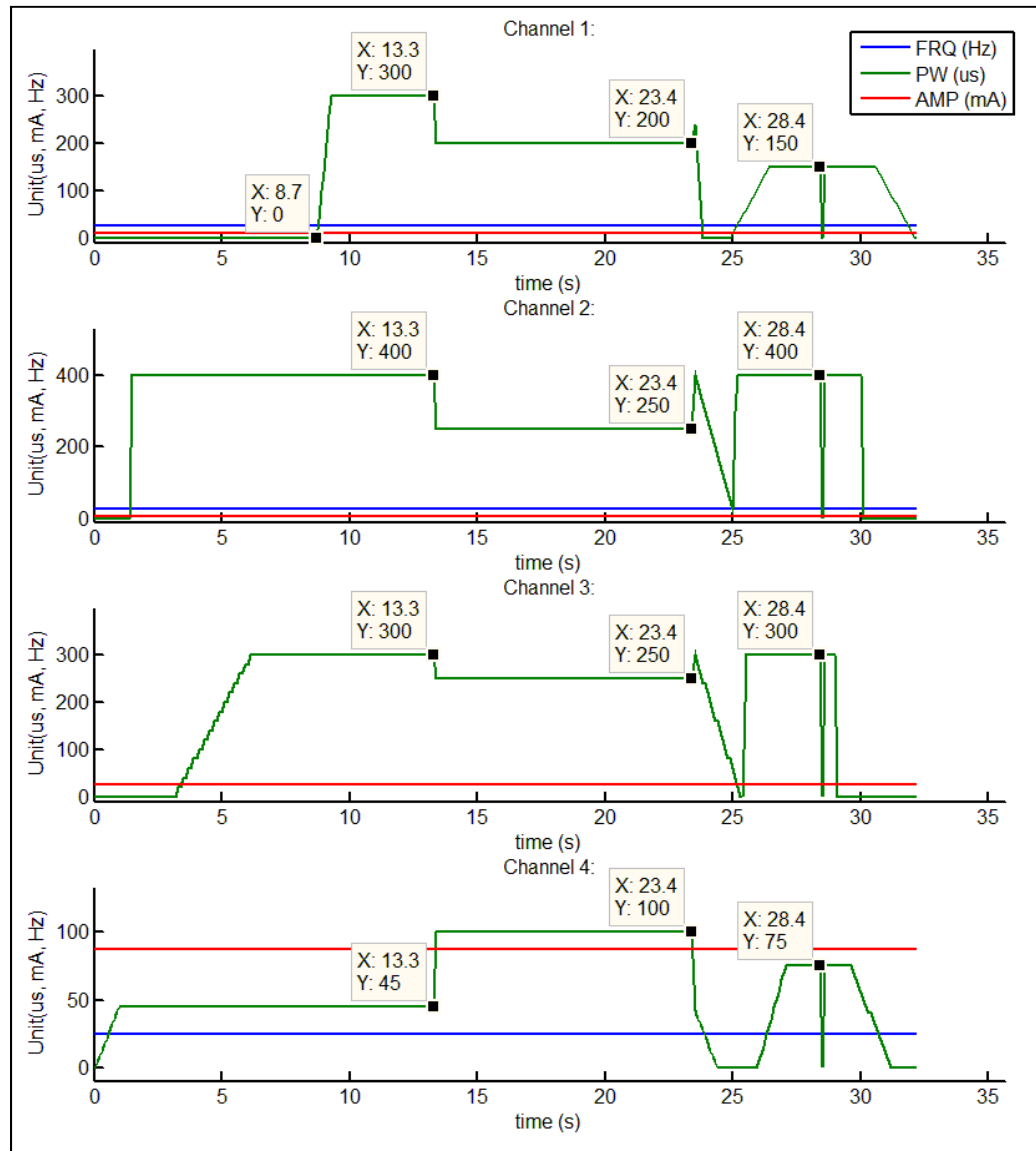


**Figure 10-10: stimProtTest04 stimulation protocol.**

The next stimulation protocol test involved the use of the push button user interaction primitives (digital input) and the channel synchronization primitive. The protocol used for this test named “stimProtTest04” is shown in Figure 10-10 above. In this test Channel 1 started off by waiting for a user controlled push button trigger (channel specific), while channels 2 to 4 started with their respective stimulation and reached the synchronous push button primitives. At this point these channels would wait until Channel 1 reached this primitive to continue. Once all channels have reached this point, the next push button input would trigger all channels to continue to their next primitives in synchronization. Then all channels would reach the TTL primitive which would synchronize the channels again and wait for another push button before continuing.

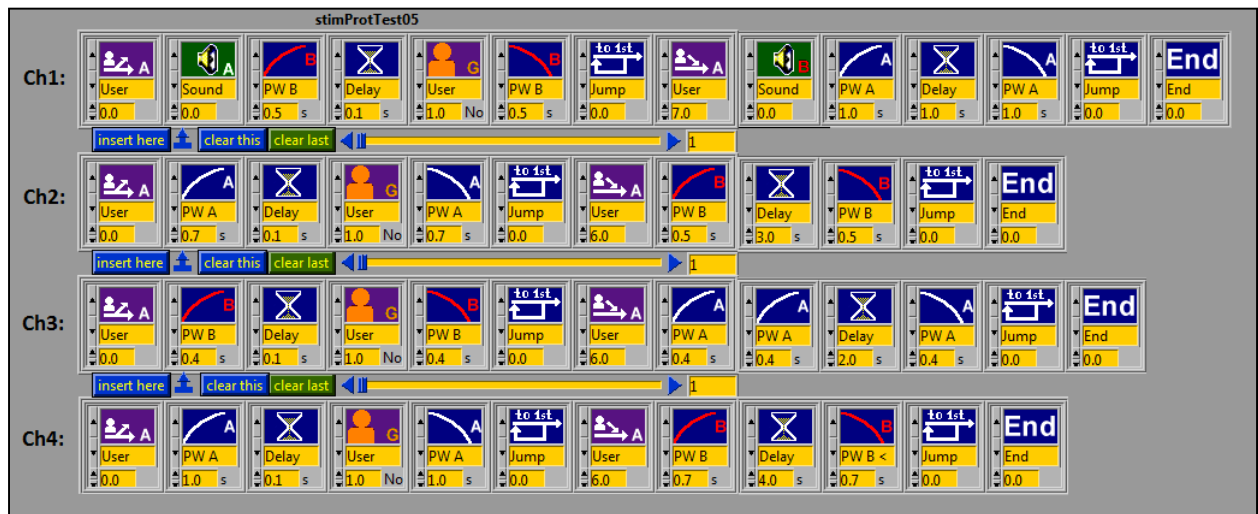
Examining Figure 10-11 below, which captures the output stimulation patterns generated by this protocol, it is evident that at 8.7 seconds Channel 1 received its first trigger to start stimulation while Channels 2 to 4 were waiting at the synchronous push button primitive. Then at 13.3 seconds the second trigger was received (sync PB), this was evident since the next primitives in each respective timelines changed the PW to a different level. Note that the constant PW

stimulation primitives in each channel were programmed to execute for different lengths of time. Then as each channel reached their corresponding synchronization primitive they continued to provide the previous constant PW until they all reach this point. Once all channels completed this primitive, the next primitives were executed synchronously; in this case the ramp down primitives in each channel were executed at 23.4 seconds.



**Figure 10-11: stimProtTest04 stimulation results.**

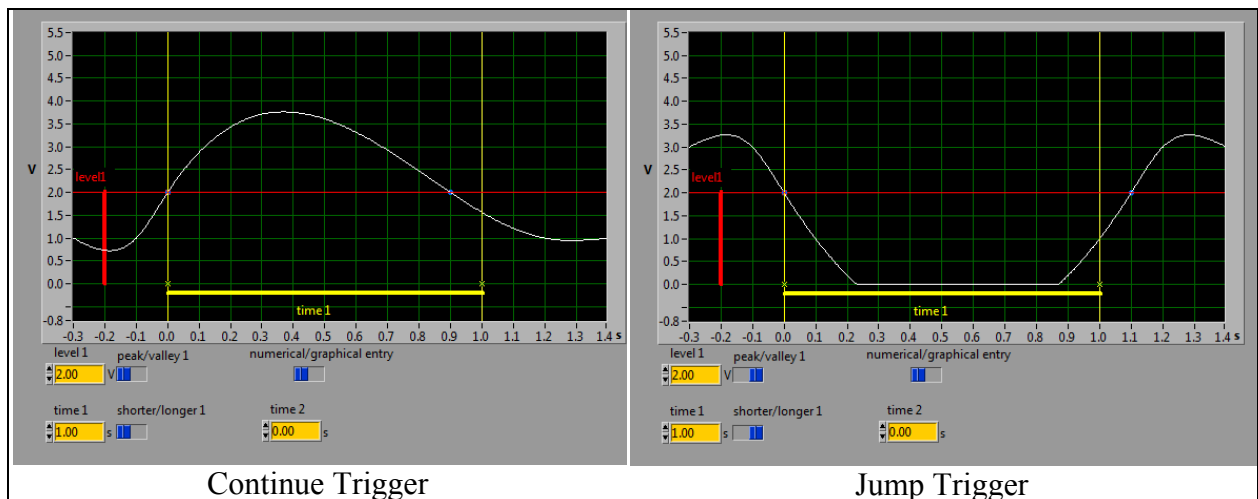
The second set of stimulation primitives in each channel executed their corresponding ramp up primitives followed by delay primitives of varying execution lengths. Then the TTL trigger was executed at different times by each channel. Once all channels reached this primitive at 28.4 seconds, the PW was throttled to zero and the controller waited for the input trigger. In this case since the TTL primitive provided the fastest response possible to a digital input trigger, the controller was executing in a tight loop waiting for this trigger and or a stop request. As it was evident from the Matlab output, there was no more data transferred to the output stage during this period. Once the trigger was received each channel immediately provided the pre-processed parameters (to the output stage, i.e., Matlab) for the next primitive and continued respective execution.



**Figure 10-12: stimProtTest05 stimulation protocol.**

The final stimulation protocol test examined the integration of the user interaction related primitives, primarily the synchronous user interaction primitives and the user branch primitives. The stimulation protocol used for this test was provided by “stimProtTest05” and is displayed in Figure 10-12 above. This protocol started off by waiting for a user interaction input to trigger

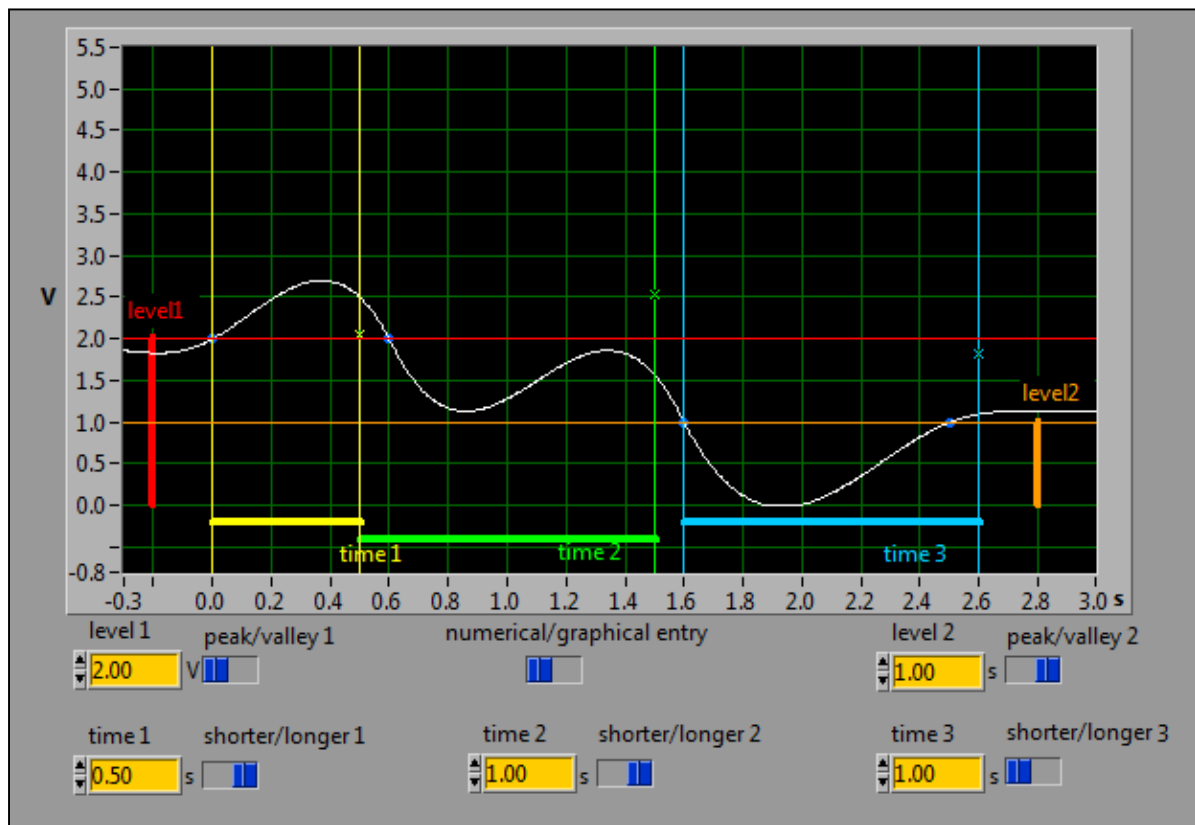
the stimulation sequence. In this case the user branch primitive was used which depending on which trigger was received (out of two possible input signals) the execution could continue with the next primitive or jump to the corresponding “mark A” primitive and continue executing from that point. In either trigger branch, the stimulation would continue and then loop back to the start where the controller would be waiting for the appropriate trigger again. The triggers for the branch primitive are shown in Figure 10-13 below. The left signal was the continue trigger and the right input signal was the jump trigger. The continue signal was triggered if the input signal from the analog sensor was greater than 2.0 V for less than 1.0 second, and the jump signal is triggered if the input signal is less than 2.0 V for more than 1.0 second. The analog input sensor used for the various user interaction triggers in this test was the potentiometer device described in Section 8.3. Depending on which branch trigger was received, a different sound primitive was incorporated within the protocol to provide an audible alert to which path had been taken.



**Figure 10-13: Branch Primitive Triggers.**

During the execution of the user branch primitive, if the continue trigger was received, the controller would continue execution until the user interaction G primitive was executed. At this

point the controller was waiting for another trigger in order to continue with the stimulation. While the system was waiting for the appropriate trigger, the PW's in each corresponding channel would be kept constant. The trigger signal for the user interaction G primitive is shown in Figure 10-14 below. Note that this is an example of a complex trigger signal, which was composed of two separate triggers that must occur with the appropriate timing. In this case, the first part of the trigger signal must be greater than 2.0 V for at least 0.5 seconds, and the second part of the trigger must arrive at least 1 second after the first part was received. The second part of the trigger must be less than 1.0 V for less than 1.0 second.

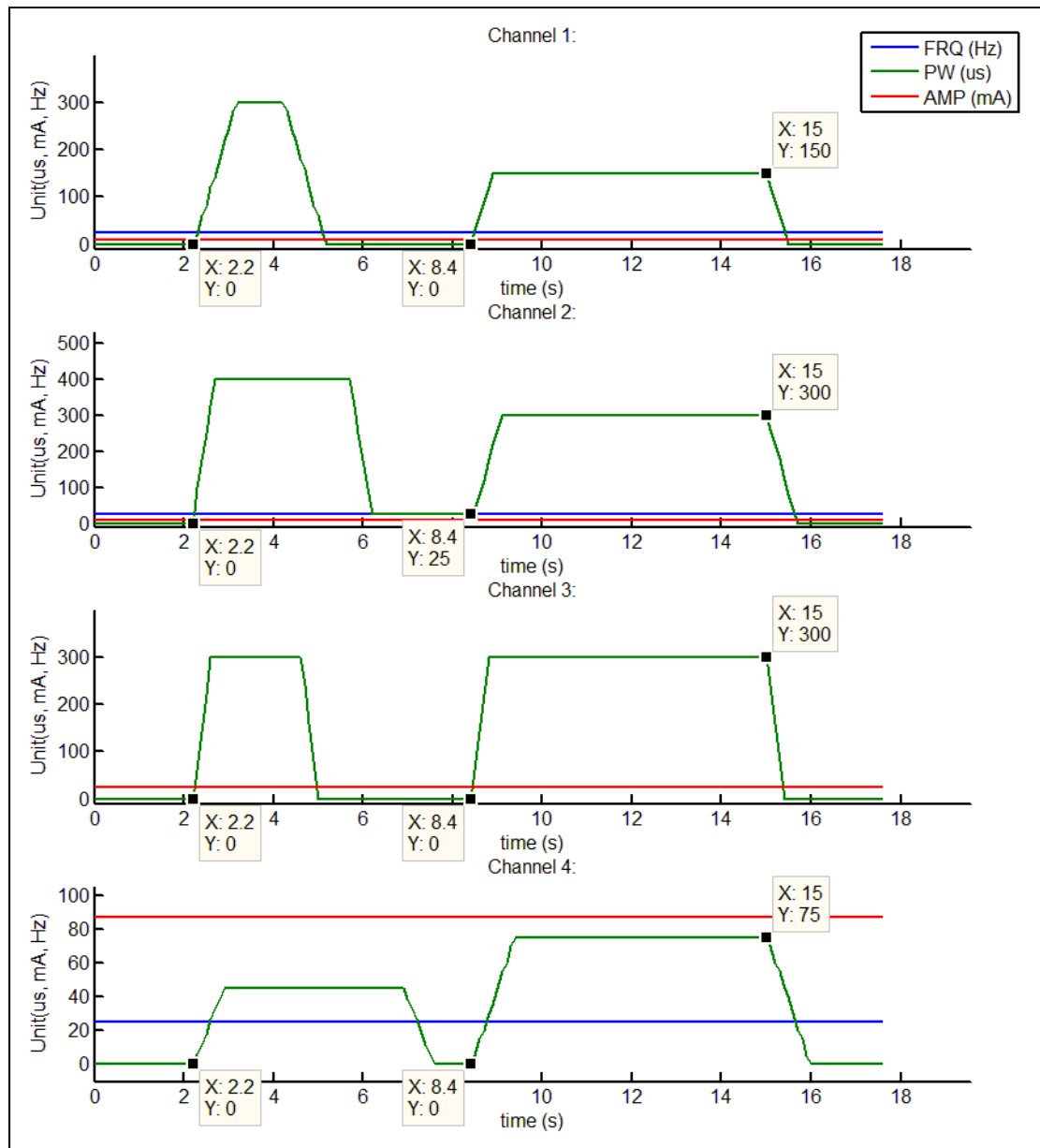


**Figure 10-14: User interaction G trigger signal.**

During the execution of this protocol, the first trigger signal provided to the branch primitive was the jump signal, which then generated the audible sound B. Then once the execution returned



back to the branch primitive, the second trigger signal provided was the continue signal, which generated the audible sound A. Finally once the stimulation returned back to the branch primitive for the third time, the stimulation was stopped by providing the stop command through the PC/Stimulator interface. Figure 10-15 below provides the stimulation output captured by Matlab for the execution of this protocol. Assessing these waveforms, it is evident that the first trigger was provided at 2.2 seconds from the start of stimulation, this trigger being the jump signal. Once the stimulation returned back to the branch primitive, the second trigger being the continue signal, was provided at 8.4 seconds. The stimulation then continued until the user interaction G primitive was executed, at this point the controller was waiting for the complex trigger described above. This trigger was received at the 15 seconds mark, at this point the ramp down primitives corresponding to each channel were executed synchronously and the stimulation was brought to a stop once the execution returned to the branch primitive for a third time.



**Figure 10-15: stimProtTest05 stimulation results.**

### 10.3.2 Stimulation Protocol Timing Testing

In this section the timing characteristics required by the controller during the decoding and execution of a stimulation protocol are examined. The main clock used by the decoding and execution structure for stimulation protocols was the peripheral timer 1 (TMR1). Details regarding its integration and configuration are provided in Section 9.3.1. The accuracy of this

timer was crucial to the operation of the controller, and therefore its configured operation had to be verified to ensure proper execution and timing accuracy. Then stress testing was carried out through the decoding and execution of a worst case stimulation protocol, this test maximized the amount of processing required by the microcontroller. The controller was then forced to a failure state in order to catch the possibility of missing a timed trigger and reporting the appropriate error. Finally the TTL fast trigger response time was measure and compared to that of the Compex Motion stimulator's documented TTL response time.

The accuracy and consistency of protocol decoding and execution clock was assessed with the use of the TMR0 peripheral timer utilized in the experimental setup. The TMR0 timer was triggered to start once the TMR1 timer was enabled for protocol decoding and execution. The timer was then stopped once the TMR1 timer triggered ten times at its timed mode operation frequency of 10 Hz (100 ms) and the value in TMR0 timer was then written through the UART serial connection to the PC. Thirty independent trials were executed measuring the accuracy and consistency of the TMR1 timer, and the results are listed in the "Stim. Trigger Timer" table of Appendix 5: Experimental Results Timing Data. The average mean time was recorded to be 1.00 seconds which corresponds to ten executions of the TMR1 timer with a period 100ms each. In each trial there was a discrepancy of an extra 1  $\mu$ s. This discrepancy was due to the minimal inaccuracies of measuring the time with the use of TMR0 timer primarily because of the 12 MHz operating frequency and the required enable/disable code added for this measurement.

Once the consistency of the stimulation decoding and execution clock had been verified, a worst case scenario protocol was created as a stress test to push the system's throughput and performance. The protocol involved initiating the parameter randomization for PW and

amplitude for each channel and the randomization of the stimulation frequency. Once all stimulation attribute randomizations were enabled, each channel had to perform a 3.2 seconds ramp up and finally, a user interaction primitive was executed by each channel. Then while the controller was waiting to capture the correct user trigger, all stimulation parameters were still being randomized at every timed sequence. Thirty separate trials were executed with this protocol and the TMR0 timer counted the amount of time between each trigger signal. This timed period was the interval where all decoding and execution of parameters took place between each trigger from the TMR1 timer. The average recorded worst case processing time for this period was 9.65 ms, see “Prot. Processing Time (Worst Case)” table in Appendix 5: Experimental Results Timing Data for the results from each trial. The worst case processing time was measured to be about one tenth of the maximum processing time before the stimulation decoding and execution was triggered for the next sequence.

During the decoding and execution of a stimulation protocol, if the processing time required for the immediate decoding and execution was greater than the trigger time, the controller would miss a trigger which could alter the stimulation output waveforms. There was an error check, which ensured that each trigger was received, if a trigger was lost, the controller would provide the appropriate error message to the user indicating a problem with the decoding and execution. During regular operations and testing this error never occurred. In order to ensure that this missed-trigger catch case was appropriately detected, the execution thread was delayed for increasing amounts of time until the error occurred. At least one trigger was missed when the induced delay was over 80 ms with the worst case protocol executing.

The final test investigated the response times associated with the TTL fast trigger. This digital trigger provided the same overall function as the synchronized push button primitive with the main difference being that the stimulation PW was brought to zero while waiting for the trigger, and the next stimulation parameters were already pre-processed and loaded, ready to be sent to the output stage on the arrival of the trigger. The end result being that once the trigger was received the output stage parameters were set immediately without waiting for the next stimulation sequence trigger (100 ms later in timed mode). The response time had been measured from the time the trigger was received and approved by the controller to the time the output stage parameters had been set. Thirty separate trials had been executed recording the systems response time to the TTL trigger and the average time was measured to be 44.6  $\mu\text{s}$ , with an error margin of 2.1  $\mu\text{s}$ . The recording from each trial are listed in table “TTL Response Time” in Appendix 5: Experimental Results Timing Data. In comparison the Compex Motion system TTL response time is documented as being less than 500  $\mu\text{s}$  [65]. The new controller subsystem provided about a tenfold improvement in TTL response performance over the original Compex Motion system.

## **11 Conclusion & Future Work**

The presented controller subsystem represents an integral component of the new stimulator developed at the Rehabilitation Engineering Laboratory. Its design and structure allow for a flexible, portable, modular and fully programmable FES system. The developed controller prototype provides optimal control and execution of custom stimulation protocols allowing the integration of various input sensors and systems for control. This system enables practitioners to easily and efficiently control the device by downloading/uploading custom stimulation protocols and controlling the stimulator through the PC/hand-held device interface. The final solution provides a low-power, efficient and fully programmable FES system controller, which can be incorporated into a novel programmable FES system.

### **11.1 Major Accomplishments for First FES Prototype**

The controller prototype developed as part of this thesis provided several improvements over previous generation FES systems in terms of capability, performance and flexibility. The final controller subsystem offers a flexible and modular design where interfaces to other systems and devices are possible. Such examples include: a) the ability to communicate with legacy Compex Motion PC software used to develop custom stimulation protocols already demonstrated in randomized control trials; b) the new generation Stimulation Control Center software used for direct control and data retrieval from controller; and c) the Matlab software used for output stage simulation during the decoding and execution of a protocol. The design provides the necessary flexibility required if additional stimulation channels or features are to be introduced.

The controller subsystem provided the necessary flexibility that allowed it to be incorporated into various FES applications, ranging from foot drop neuroprostheses to reaching and grasping neuroprostheses, to standing and walking neuroprostheses, without requiring re-design or development of the base hardware and software. Such a wide range of applications make this system especially advantageous for use in SCI rehabilitation clinics where the various injuries would require specialized FES neuroprostheses. Using the same base hardware platform, the system can be transformed from one application to another in seconds by simply flashing the appropriate protocol. Clinicians can also use the Stimulation Control Center software to directly control the stimulator from its GUI and access or alter immediate stimulation specific data in microseconds.

## **11.2 Areas of Extension and Future Work**

The initial design of the controller subsystem focused on a modular and flexible solution with anticipation into possible areas of expansion that would make possible for future extensions and capabilities without requiring a re-design and development of a new system. There are numerous possible areas of extension and future work that can be added to the base design. Some examples include the addition of stimulation channels with a modular output stage interface, simulation of stimulation protocol support, extensions of the Stimulation Control Center software, non-volatile flash memory storage for stimulation protocols and related data, and implantable FES specialization. Another area of immediate future work is the packaging and development of the driver board which includes all the components of the new stimulator system. The main components comprise of the controller, output stage, user interface, PC/Hand-held device interface, and the power/battery interface. Once the overall stimulator has been integrated, an overall system cost and usability analysis can be carried out.

A modular output stage interface will allow for additional stimulation channels based on the requirements of the intended neuroprosthesis. The current base design supports four stimulation channels that can be easily extended. As part of the future work, the output stage interface can be designed to be a separate physical component that can be integrated by simply plugging into the controller subsystem. This methodology would allow the development of different output stages with various number of stimulation channels, for example a two, four, eight and ten channel output-stage. Then depending what is required, the appropriate output stage can be connected to the system. The controller would be compatible with any range of the supported output stages.

The Matlab software developed during the implementation of the controller subsystem was utilized primarily to receive the data intended for the output stage component of the FES system. This idea can be extended into a simulation software suite that would allow practitioners to test their protocol to ensure proper stimulation patterns will be generated prior to deploying the neuroprosthesis. This extension would allow simulation of the output stage such that stimulation patterns on each channel can be visually confirmed and tested during the development of a stimulation protocol without requiring additional hardware to view the stimulation waveforms directly from the physical output stage.

The Stimulation Control Center software has been designed to interface the stimulator to manage the execution of arbitrary stimulation protocols. This includes visualizing and altering the various stimulation parameters during stimulation. This software can be extended and ported to a hand-held device to allow for portable access to stimulator systems. Since the interface to the stimulator is through USB, the hand-held device can also be used to provide charge to the



stimulator if required. If non-volatile memory is added to the controller through the SD/MMC interface on the microcontroller, the PC/hand-held device software can be extended to select the various stimulation protocols stored on the flash memory device. This would allow the HCP to quickly swap stimulation protocols without having to re-download them repeatedly.

The base controller subsystem has been designed primarily for surface stimulation FES systems, but due to the flexible decoding and execution structure this design can be extended for future implantable neuroprosthetic FES systems. The main difference between surface and implantable FES systems is the range of parameter values supported by the output stage. In an implantable FES system, the stimulation electrodes have direct access to the muscles and nerves and therefore require considerably less current to inhibit the desired function in comparison to a surface FES systems. The majority of the re-design and development of such a system would be concentrated in the output stage component since performance and miniaturization are key.

## References

- [1] P. Milos R. Popovic, P.Eng. and P. T. Adam Thrasher, "How engineers are helping injured people walk again", *The Journal of Policy Engagement*, vol. 1, 2009.
- [2] M. R. Popovic and T. A. Thrasher, "Neuroprostheses", *Encyclopedia of biomaterials and biomedical engineering*, vol. 3, pp. 1924-1933, 2004.
- [3] ualberta, (2010, 01-11), *BIONIC GLOVE*, Available: <http://www.ualberta.ca/~aprochaz/bgtemp.html>
- [4] Bioness, (2010, 01-11), *NESS H200 Hand Rehabilitation System*, Available: [http://www.bioness.com/NESS\\_H200\\_for\\_Hand\\_Rehab.php](http://www.bioness.com/NESS_H200_for_Hand_Rehab.php)
- [5] O. M. Limited, (2010, 01-11), *ODFS® PACE Stimulator*, Available: <http://www.odstockmedical.com/patientssite/ODFSpace.html>
- [6] Bioness, (2010, 01-11), *NESS L300 Foot Drop System*, Available: [http://www.bioness.com/NESS\\_L300\\_for\\_Foot\\_Drop.php](http://www.bioness.com/NESS_L300_for_Foot_Drop.php)
- [7] Walkaide, (2010, 01-11), *The WalkAdie System For Treatment of Foot Drop*, Available: <http://www.walkaide.com/en-US/Pages/default.aspx>
- [8] Parastep, (2010, 01-11), *The Parastep System*, Available: <http://www.musclepower.com/parastep.htm>
- [9] EMPI, (2010, 01-11), *300PV™ Portable Neuromuscular Stimulation Device*, Available: [http://www.empi.com/empi\\_products/detail.aspx?id=188](http://www.empi.com/empi_products/detail.aspx?id=188)
- [10] Medel, (2010, 01-11), *MOTIONSTIM 8*, Available: <http://www.medel-hamburg.de/>
- [11] CefarCompex, (2010, 01-11), *Compex3 Professional Muscle Stimulator*, Available: [http://www.cefarcampex.com/en\\_EU/Compex-3.html](http://www.cefarcampex.com/en_EU/Compex-3.html)
- [12] Aetna, (2000, 01-11), *Clinical Policy Bulletin: NeuroControl Freehand System*, Available: [http://www.aetna.com/cpb/medical/data/300\\_399/0378.html](http://www.aetna.com/cpb/medical/data/300_399/0378.html)
- [13] Synapse, (2010, 01-11), *The NeuRx DPST™* Available: <http://www.synapsebiomedical.com/products/neurx.shtml>
- [14] "Heart and Stroke Foundation of Canada", <http://www.heartandstroke.com/site/c.ikIQLcMWJtE/b.3483991/k.3448/Statistics.htm#stroke>, 2009.
- [15] "International Collaboration On Repair Discoveries (ICORD)", <http://www.icord.org/sci.html>, 2009.

- [16] S. Hamid and R. Hayek, "Role of electrical stimulation for rehabilitation and regeneration after spinal cord injury: an overview", *European Spine Journal*, vol. 17, pp. 1256-1269, Sep 2008.
- [17] K. T. Ragnarsson, "Functional electrical stimulation after spinal cord injury: current use, therapeutic effects and future directions", *Spinal Cord*, vol. 46, pp. 255-74, Apr 2008.
- [18] T. A. Thrasher, *et al.*, "Rehabilitation of reaching and grasping function in severe hemiplegic patients using functional electrical stimulation therapy", *Neurorehabil Neural Repair*, vol. 22, pp. 706-14, Nov-Dec 2008.
- [19] Y. Hara, *et al.*, "A home-based rehabilitation program for the hemiplegic upper extremity by power-assisted functional electrical stimulation", *Disabil Rehabil*, vol. 30, pp. 296-304, 2008.
- [20] I. Morita, *et al.*, "Reconstruction of upper limb motor function using functional electrical stimulation (FES)", *Acta neurochirurgica. Supplement*, vol. 97, pp. 403-407, 2007.
- [21] G. Alon, *et al.*, "Functional electrical stimulation enhancement of upper extremity functional recovery during stroke rehabilitation: a pilot study", *Neurorehabil Neural Repair*, vol. 21, pp. 207-15, May-Jun 2007.
- [22] W. K. Durfee, "Gait Restoration by Functional Electrical Stimulation", *Department of Mechanical Engineering, University of Minnesota, Minneapolis, USA wkdurfee@umn.edu*, 2006.
- [23] S. Mangold, *et al.*, "Transcutaneous functional electrical stimulation for grasping in subjects with cervical spinal cord injury", *Spinal Cord*, vol. 43, pp. 1-13, Jan 2005.
- [24] M. R. Popovic, *et al.*, "Neuroprostheses for grasping", *Neurol Res*, vol. 24, pp. 443-52, Jul 2002.
- [25] M. R. Popovic, *et al.*, "Surface-stimulation technology for grasping and walking neuroprostheses", *Engineering in Medicine and Biology Magazine, IEEE*, vol. 20, pp. 82-93, 2001.
- [26] M. H. Granat, *et al.*, "The Role of Functional Electrical-Stimulation in the Rehabilitation of Patients with Incomplete Spinal-Cord Injury - Observed Benefits during Gait Studies", *Paraplegia*, vol. 31, pp. 207-215, Apr 1993.
- [27] R. Turk and P. Obreza, "Functional electrical stimulation as an orthotic means for the rehabilitation of paraplegic patients", *Paraplegia*, vol. 23, pp. 344-8, Dec 1985.
- [28] T. A. Thrasher and M. R. Popovic, "Functional electrical stimulation of walking: function, exercise and rehabilitation", *Ann Readapt Med Phys*, vol. 51, pp. 452-60, Jul 2008.
- [29] M. R. Popovic, "FES Therapy for Improving Grasping in Individuals After SCI & Brain Machine Interfaces", in *Dr. Jousse Lecture Series – September 2009 ed*, 2009.

- [30] L. Baker, *et al.*, "NeuroMuscular Electrical Stimulation - a Practical Guide", 4th ed. Downey California, USA: Los Amigos Research & Education Institute, p. 251, 2000.
- [31] H. Fodstad and M. Hariz, "Electricity in the treatment of nervous system disease", *Acta Neurochir Suppl*, vol. 97, pp. 11-9, 2007.
- [32] U. Oh, "Historical Evolution of Pain Concepts", in *The Nociceptive Membrane*, Amsterdam : Elsevier, Ed., ed, 2006, pp. 5-9.
- [33] W. T. Liberson, *et al.*, "Functional electrotherapy: stimulation of the peroneal nerve synchronized with the swing phase of the gait of hemiplegic patients", *Arch Phys Med Rehabil*, vol. 42, pp. 101-5, Feb 1961.
- [34] F. S. Grodins, *et al.*, "Stimulation of denervated skeletal muscle with alternating current", *American Journal of Physiology*, vol. 142, pp. 0216-0221, Sep 1944.
- [35] D. Graupe, *et al.*, "Stochastically-modulated stimulation to slow down muscle fatigue at stimulated sites in paraplegics using functional electrical stimulation for leg extension", *Neurol Res*, vol. 22, pp. 703-4, Oct 2000.
- [36] M. R. Popovic and T. Keller, "Modular transcutaneous functional electrical stimulation system", *Medical Engineering & Physics*, vol. 27, pp. 81-92, 2004.
- [37] J. P. McCabe, *et al.*, "Feasibility of combining gait robot and multichannel functional electrical stimulation with intramuscular electrodes", *J Rehabil Res Dev*, vol. 45, pp. 997-1006, 2008.
- [38] A. Prochazka, *et al.*, "The Bionic Glove: An electrical stimulator garment that provides controlled grasp and hand opening in quadriplegia", *Archives of Physical Medicine and Rehabilitation*, vol. 78, pp. 608-614, Jun 1997.
- [39] N. Negard, *et al.*, "Application Programming Interface and PC control for the 8 channel stimulator MOTIONSTIM8", *10th Annual Conference of the International FES Society*, vol. 10, 2005.
- [40] P. H. Peckham and J. S. Knutson, "Functional electrical stimulation for neuromuscular applications", *Annual Review of Biomedical Engineering*, vol. 7, pp. 327-360, 2005.
- [41] J. Knutson, *et al.*, "Interventions for Mobility and Manipulation After Spinal Cord Injury: A Review of Orthotic and Neuroprosthetic Options", *Topics in Spinal Cord Injury Rehabilitation*, vol. 11, pp. 61-81, 2006.
- [42] P. H. Peckham, *et al.*, "Efficacy of an implanted neuroprosthesis for restoring hand grasp in tetraplegia: A multicenter study", *Archives of Physical Medicine and Rehabilitation*, vol. 82, pp. 1380-1388, Oct 2001.

- [43] D. M. R. Popovic and D. T. Keller, *Complex Motion: Neuroprosthesis for grasping applications*: Enabling Technologies: Body Image and Body Function, 2002.
- [44] O. M. Group, "About the Object Management Group™ (OMG™)", 2010.
- [45] S. S. Alhir, "Understanding the Unified Modeling Language (UML)", *Methods & Tools*, vol. 7, pp. 11-18, 1999.
- [46] V. Kakkar, "ARM BASED ARCHITECTURE FOR COCHLEAR IMPLANT", *Ubiquitous Computing and Communication Journal*.
- [47] ARM7™, "ARM Cortex-M3 Processor Software Development for ARM7TDMI Processor Programmers", July 2009.
- [48] NXP-Semiconductors, (2010, 01-11), *LPC23xx series device highlight*, Available: <http://ics.nxp.com/products/lpc2000/lpc23xx/>
- [49] Keil, (2010, 01-11), *MCB2300 Evaluation Board*, Available: <http://www.keil.com/mcb2300/>
- [50] Keil, "RTX Memory Requirements", [http://www.keil.com/arm/rl-arm/rtx\\_size.asp](http://www.keil.com/arm/rl-arm/rtx_size.asp), 2010.
- [51] NXP-Semiconductors, "UM10211 LPC23XX User manual", 2009.
- [52] R. Smith, (2010, 01-11), *QUICK REFERENCE FOR RS485, RS422, RS232 AND RS423*, Available: <http://www.rs485.com/rs485spec.html>
- [53] N. Instruments, (2008, 01-11), *USB for Automated Test*, Available: <http://zone.ni.com/devzone/cda/tut/p/id/7278>
- [54] "Universal Serial Bus Class Definitions for Communication Devices", vol. 1.1, 1999.
- [55] B. SIG, (2010, 01-11), *Bluetooth Basics*, Available: <http://www.bluetooth.com/English/Technology/Pages/Basics.aspx>
- [56] "Sena Technologies White Paper: Latency/Throughput Test of Bluetooth-Serial Adapters", Sena Technologies, 2008.
- [57] MeshNetics, (2009, 01-11), *ZigBee FAQ*, Available: <http://www.meshnetics.com/zigbee-faq/#16>
- [58] M. G. Dekenah, (2004, 01-11), *RS232 'Sniffer' Probe*, Available: <http://www.marcspages.co.uk/tech/3104.htm>
- [59] M. Barr, "Additive Checksums", *Embedded Systems Programming*, 1999.
- [60] Keil, (2008, 01-11), *Keil MCB2300 Schematics (47 ed.)*, Available: <http://www.keil.com/mcb2300/mcb2300-schematics.pdf>

- [61] R. B. S. I. M. Rongching Dai, Brian J. Andrews IEEE Member, Kelvin B. James, and Marguerite Wieler, "Application of Tilt Sensors in Functional Electrical Stimulation", *IEEE Transactions on Rehabilitation Engineering*, vol. 4, pp. 63-72, 1996.
- [62] A. Kostov, *et al.*, "Machine learning in control of functional electrical stimulation systems for locomotion", *IEEE Trans Biomed Eng*, vol. 42, pp. 541-51, Jun 1995.
- [63] E. Express, "Contact Bounce and De-Bouncing", [http://www.elexp.com/t\\_bounc.htm](http://www.elexp.com/t_bounc.htm), 2010.
- [64] K. W. M. Stephen K. Park, "Random Number Generators: Good Ones are Hard to Find", *Computing Practices*, 1988.
- [65] T. Keller, "Surface Functional Electrical Stimulation (FES) Neuroprostheses for Grasping", 2001.

## Appendix 1: Requirements Models

Scenario	Description
Download stimulation protocol	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller for transmission of stimulation protocol to stimulator.</li> <li>Controller receives and stores stimulation protocol.</li> </ul>
Upload stimulation protocol	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller for transmission of stimulation protocol from stimulator.</li> <li>Controller sends requested stimulation protocol.</li> </ul>
Start stimulation	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller with request to start stimulation or user instantiates start directly.</li> <li>Controller initializes the stimulator and starts the stimulation (assuming valid stimulation protocol).</li> </ul>
Stop stimulation	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller with request to stop stimulation or user instantiates stop directly.</li> <li>Controller stops stimulation by bringing the amplitude and PW parameters for each stimulation channel to zero.</li> </ul>
Pause stimulation	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller with request to pause stimulation.</li> <li>Controller brings the amplitude and PW parameters for each stimulation channel to zero.</li> <li>Previous state within stimulation protocol is saved.</li> </ul>
Resume stimulation	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller with request to resume stimulation.</li> <li>Controller loads saved state within stimulation protocol and continues its execution.</li> </ul>
Adjust stimulation parameters	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller with request to change a stimulation parameter.</li> <li>Controller makes the appropriate parameter change within the ongoing (i.e., presently administered) stimulation protocol.</li> </ul>
Adjust stimulation protocol	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller with request to adjust the stimulation protocol.</li> <li>Controller makes appropriate change(s) within the current stimulation protocol.</li> </ul>
Process stimulation protocol	<ul style="list-style-type: none"> <li>With timed execution, the controller processes the next stimulation parameters from loaded stimulation protocol.</li> <li>The final parameters for each stimulation channel are set for the output stage for each time instance within the stimulation.</li> </ul>
Capture data	<ul style="list-style-type: none"> <li>PC/hand-held device initiates connection to controller with request to transmit desired data (e.g., stimulator status, current stimulation parameters, sensor data, stimulation electrode data, etc.).</li> <li>Controller packages requested data and transmits it back to the PC/hand-held device.</li> </ul>
Sensor or electrode failure	<ul style="list-style-type: none"> <li>Controller detects failure of specific sensor(s) or electrode(s).</li> <li>If electrode failure: stimulation is stopped – safety.</li> <li>If sensor fails: stimulation is stopped depending on its use – e.g., control stimulation amplitude.</li> <li>Appropriate message is displayed and status is updated.</li> </ul>
Battery charge in progress	<ul style="list-style-type: none"> <li>Controller stops any current running stimulation during charging period.</li> </ul>
Battery fully charged	<ul style="list-style-type: none"> <li>Controller displays appropriate message and goes into sleep or idle mode.</li> </ul>

**Table A1-1: Complete list of fundamental scenarios**

<b>Use case name</b>	Download stimulation protocol
<b>Primary actor(s)</b>	PC/hand-held device
<b>Secondary Actor(s)</b>	HCP/Patient/Administrator/Engineer – user
<b>Pre-condition</b>	1. The PC/hand-held device establishes a connection to download a stimulation protocol to the stimulator.
<b>Flow of events</b>	2. The controller acknowledges a valid connection and proceeds with the request. 3. The PC/hand-held device transmits entire stimulation protocol to controller. 4. The controller validates and stores the stimulation protocol.
<b>Post-condition</b>	5. The controller updates its status and displays an appropriate message.
<b>Alternate Flows</b>	2.3 Current stimulation running. 2.4 The controller rejects the request from the PC/hand-held device. 3.3 Error detected in received stimulation program file. 3.4 The controller updates its status and displays an appropriate error message for the user.
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	The appropriate connection is made to the PC/hand-held device.

**Table A1-2: Download stimulation protocol use case**

<b>Use case name</b>	Upload stimulation protocol
<b>Primary actor(s)</b>	PC (personal computer), hand-held device
<b>Secondary Actor(s)</b>	HCP/Patient/Administrator/Engineer – user
<b>Pre-condition</b>	1. The PC/hand-held device initiates request to upload the current stimulation protocol from the stimulator.
<b>Flow of events</b>	2. The controller acknowledges a valid connection and proceeds with the request. 3. The controller transmits current stimulation protocol to the PC/hand-held device. 4. The PC/hand-held device acknowledges successful transfer.
<b>Post-condition</b>	5. The controller displays an appropriate message.
<b>Alternate Flows</b>	2.1 No stimulation protocol on stimulator. 2.2 The controller rejects the request from the PC/hand-held device. 3.1 Error detected in sent stimulation program file by PC/hand-held device. 3.2 The controller displays appropriate error message for the user.
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	<ul style="list-style-type: none"> <li>PC/hand-held device is connected to stimulator.</li> </ul>

**Table A1-3: Upload stimulation protocol use case**



<b>Use case name</b>	Start Stimulation
<b>Primary actor(s)</b>	HCP/Patient/Administrator/Engineer – user or PC/hand-held device
<b>Secondary Actor(s)</b>	Output Stage
<b>Pre-condition</b>	1. The user or PC/hand-held device has initiated the start of the stimulation.
<b>Flow of events</b>	2. The controller initializes stimulation parameters based on protocol and configures the output stage 3. Stimulation status is updated and this information is display on the stimulator
<b>Post-condition</b>	4. Stimulation sequence execution has started.
<b>Alternate Flows</b>	2.4 Stimulation program file is not loaded in memory. 2.5 Controller updates status and displays error message for user. 2.6 Stimulation is not started. 2.4 Stimulation already running 2.5 If the command arrives from the stimulator – this indicates a stop 2.6 If the command arrives from the PC/hand-held device – the stimulation continues
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	The controller is in idle or ready mode.

**Table A1-4: Start stimulation use case**

<b>Use case name</b>	Stop Stimulation
<b>Primary actor(s)</b>	HCP/Patient/Administrator/Engineer – user or PC/hand-held device
<b>Secondary Actor(s)</b>	Output Stage
<b>Pre-condition</b>	1. The user or PC/hand-held device initiates request to stop stimulation.
<b>Flow of events</b>	2. The controller brings the amplitude and PW waveform parameters to zero for each stimulation channel. 3. The current stimulation protocol is deactivated from further processing and the controller returns to idle state. 4. The controller updates its status and displays an appropriate message.
<b>Post-condition</b>	5. Current stimulation is stopped.
<b>Alternate Flows</b>	2.1 Stimulation already stopped or not running. 2.2 If requested from PC/hand-held device, request is ignored. If requested directly from stimulator interface, this indicates a start of stimulation.
<b>Non-functional requirements</b>	<ul style="list-style-type: none"> <li>Stop command from stimulator unit is treated in hard real-time manner.</li> </ul>
<b>Assumptions</b>	N/A

**Table A1-5: Stop stimulation use case**

<b>Use case name</b>	Pause Stimulation
<b>Primary actor(s)</b>	PC/hand-held device
<b>Secondary Actor(s)</b>	Output Stage
<b>Pre-condition</b>	1. The PC/hand-held device initiates request to pause stimulation.
<b>Flow of events</b>	2. The controller brings the amplitude and PW waveform parameters to zero for each stimulation channel. 3. Current state of stimulation within stimulation protocol is saved. 4. The controller updates its status and displays an appropriate message.
<b>Post-condition</b>	5. Current stimulation is paused.
<b>Alternate Flows</b>	2.1 Stimulation already stopped or not running 2.2 The controller ignores request.
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	<ul style="list-style-type: none"> <li>PC/hand-held device is connected to stimulator.</li> </ul>

**Table A1-6: Pause stimulation use case**

<b>Use case name</b>	Resume Stimulation
<b>Primary actor(s)</b>	PC/hand-held device
<b>Secondary Actor(s)</b>	Output Stage
<b>Pre-condition</b>	1. The PC/hand-held device initiates request to resume stimulation.
<b>Flow of events</b>	2. The controller loads saved stimulation state. 3. The controller updates its status and displays an appropriate message. 4. Controller continues processing stimulation protocol (Refer to Process Stimulation Protocol use case).
<b>Post-condition</b>	5. Current stimulation is resumed.
<b>Alternate Flows</b>	2.1 Stimulation is already running 2.2 The controller ignores request.
<b>Non-functional requirements</b>	<ul style="list-style-type: none"> <li>Stimulation Intensity (amplitude) is gradually resumed (linear ramp) over a fixed period of time of 1 sec</li> </ul>
<b>Assumptions</b>	<ul style="list-style-type: none"> <li>PC/hand-held device is connected to stimulator.</li> </ul>

**Table A1-7: Resume stimulation use case**

<b>Use case name</b>	Adjust Stimulation Parameters
<b>Primary actor(s)</b>	PC/hand-held device
<b>Secondary Actor(s)</b>	N/A
<b>Pre-condition</b>	1. PC/hand-held device initiates request to adjust a current stimulation parameter.
<b>Flow of events</b>	2. The controller receives parameter change request and makes the necessary changes.
<b>Post-condition</b>	3. Stimulation parameter has been changed.
<b>Alternate Flows</b>	2.1 There is no active stimulation 2.2 Controller ignores request 2.1 Parameter change exceeds safety margin 2.2 Controller keep parameter within safety margin
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	<ul style="list-style-type: none"> <li>PC/hand-held device is connected to stimulator.</li> <li>Controller is currently processing and executing a stimulation protocol</li> </ul>

**Table A1-8: Adjust stimulation parameters use case**

<b>Use case name</b>	Adjust Stimulation Protocol
<b>Primary actor(s)</b>	PC/hand-held device
<b>Secondary Actor(s)</b>	N/A
<b>Pre-condition</b>	1. PC/hand-held device initiates request to adjust current stimulation protocol.
<b>Flow of events</b>	2. Controller receives and validates change request 3. Changes are applied to the stimulation protocol
<b>Post-condition</b>	4. Stimulation protocol is updated with appropriate changes.
<b>Alternate Flows</b>	3.1 There is no stimulation protocol currently loaded in stimulator 3.2 Controller ignores request.
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	<ul style="list-style-type: none"> <li>PC/hand-held device is connected to stimulator.</li> <li>Valid stimulation protocol is loaded</li> </ul>

**Table A1-9: Adjust stimulation protocol use case**

<b>Use case name</b>	Process Stimulation Protocol
<b>Primary actor(s)</b>	PC/hand-held device
<b>Secondary Actor(s)</b>	Input sensors, Display, speaker.
<b>Pre-condition</b>	1. Valid stimulation protocol is loaded and a start command has been issued.
<b>Flow of events</b>	2. The controller processes the stimulation protocol in a real-time manner and continuously controls the output stage to generate the desired stimulation waveforms (refer to stimulation programmability support in Table 4-3 for type of processing functionality). 3. Depending on stimulation protocol, the controller can receive input sensor data for triggering and control of the stimulation. It can also display messages for pre-defined periods of time as well as play sounds at key instances within stimulation.
<b>Post-condition</b>	4. Controller continues processing stimulation protocol.
<b>Alternate Flows</b>	N/A
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	Valid stimulation protocol is loaded in stimulator.

**Table A1-10: Process stimulation protocol use case**

<b>Use case name</b>	Captured Data
<b>Primary actor(s)</b>	PC/hand-held device
<b>Secondary Actor(s)</b>	Input Sensors
<b>Pre-condition</b>	1. PC/hand-held device initiates request to receive data from controller.
<b>Flow of events</b>	2. Requested data is processed and returned to PC/hand-held device. This can include: stimulator status, current stimulation parameters, input sensor data and stimulation electrode data.
<b>Post-condition</b>	3. Requested data is sent to PC/hand-held device
<b>Alternate Flows</b>	2.1 The specific request for data is invalid 2.2 Controller ignores request
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	PC/hand-held device is connected to stimulator.

**Table A1-11: Capture data use case**

<b>Use case name</b>	Sensor or electrode failure
<b>Primary actor(s)</b>	Battery
<b>Secondary Actor(s)</b>	HCP/Patient/Administrator/Engineer – user
<b>Pre-condition</b>	1. A sensor or electrode failure is detected.
<b>Flow of events</b>	2. If an electrode failed, controller will stop stimulation and bring amplitudes and PWs for each channel to zero. If a sensor failed it will do the same if it is being used to trigger or control stimulation.
<b>Post-condition</b>	3. The controller updates its status and displays an appropriate message.
<b>Alternate Flows</b>	N/A
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	Stimulation is currently running.

**Table A1-12: Sensor or electrode failure use case**

<b>Use case name</b>	Battery charge in progress
<b>Primary actor(s)</b>	Battery
<b>Secondary Actor(s)</b>	HCP/Patient/Administrator/Engineer – user
<b>Pre-condition</b>	1. Detection of battery charge in progress.
<b>Flow of events</b>	2. Controller stops any active stimulation and brings amplitudes and PWs for each channel to zero.
<b>Post-condition</b>	3. The controller updates its status and displays an appropriate message.
<b>Alternate Flows</b>	N/A
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	The stimulator has been connected to power source.

**Table A1-13: Battery charge in progress use case**

<b>Use case name</b>	Battery fully charged
<b>Primary actor(s)</b>	Battery
<b>Secondary Actor(s)</b>	HCP/Patient/Administrator/Engineer – user
<b>Pre-condition</b>	1. Battery level reaches full capacity.
<b>Flow of events</b>	2. The controller updates its status and displays an appropriate message.
<b>Post-condition</b>	3. Controller goes into sleep or idle mode.
<b>Alternate Flows</b>	N/A
<b>Non-functional requirements</b>	N/A
<b>Assumptions</b>	N/A

**Table A1-14: Battery fully charged use case**

## Appendix 2: Static System Model

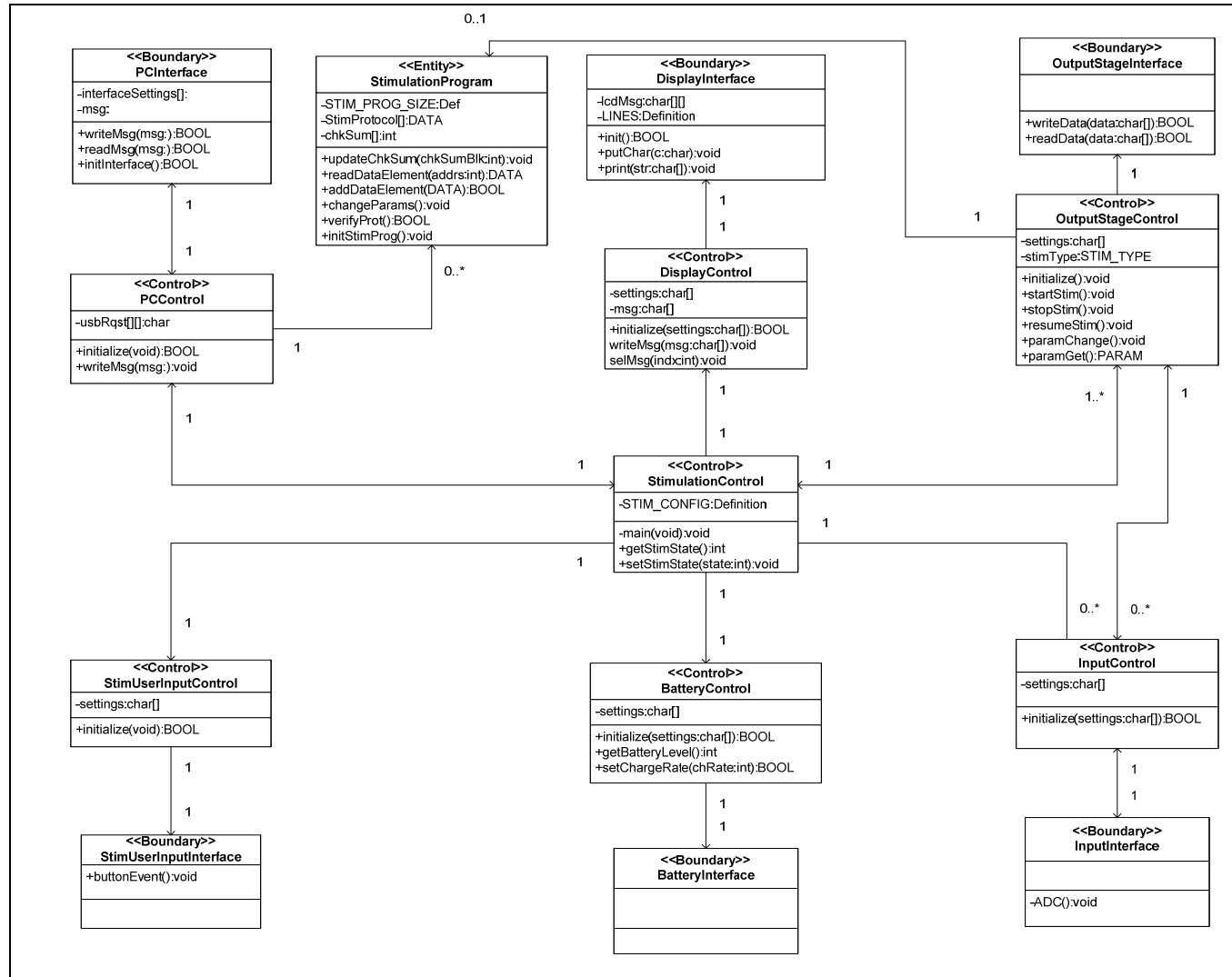
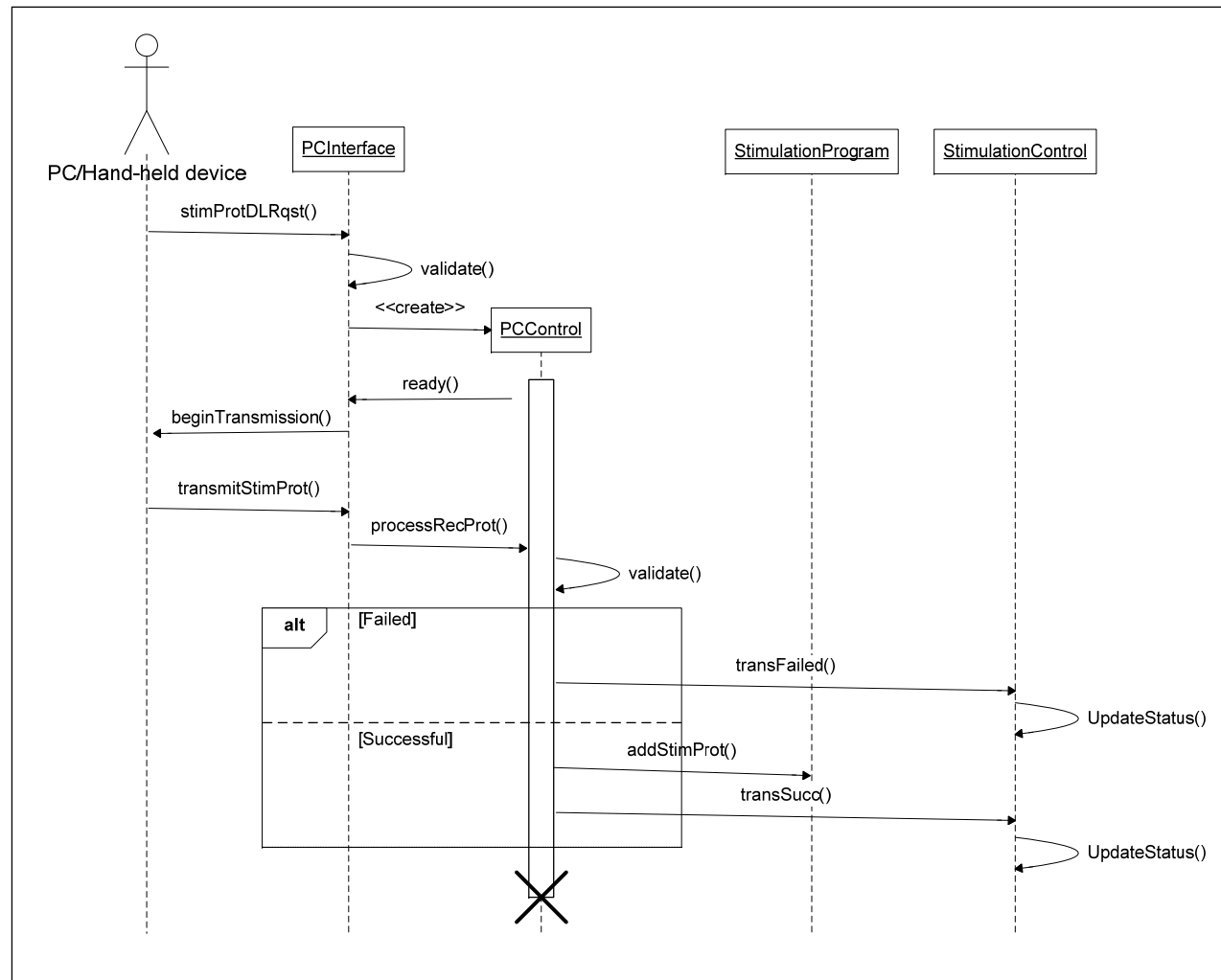
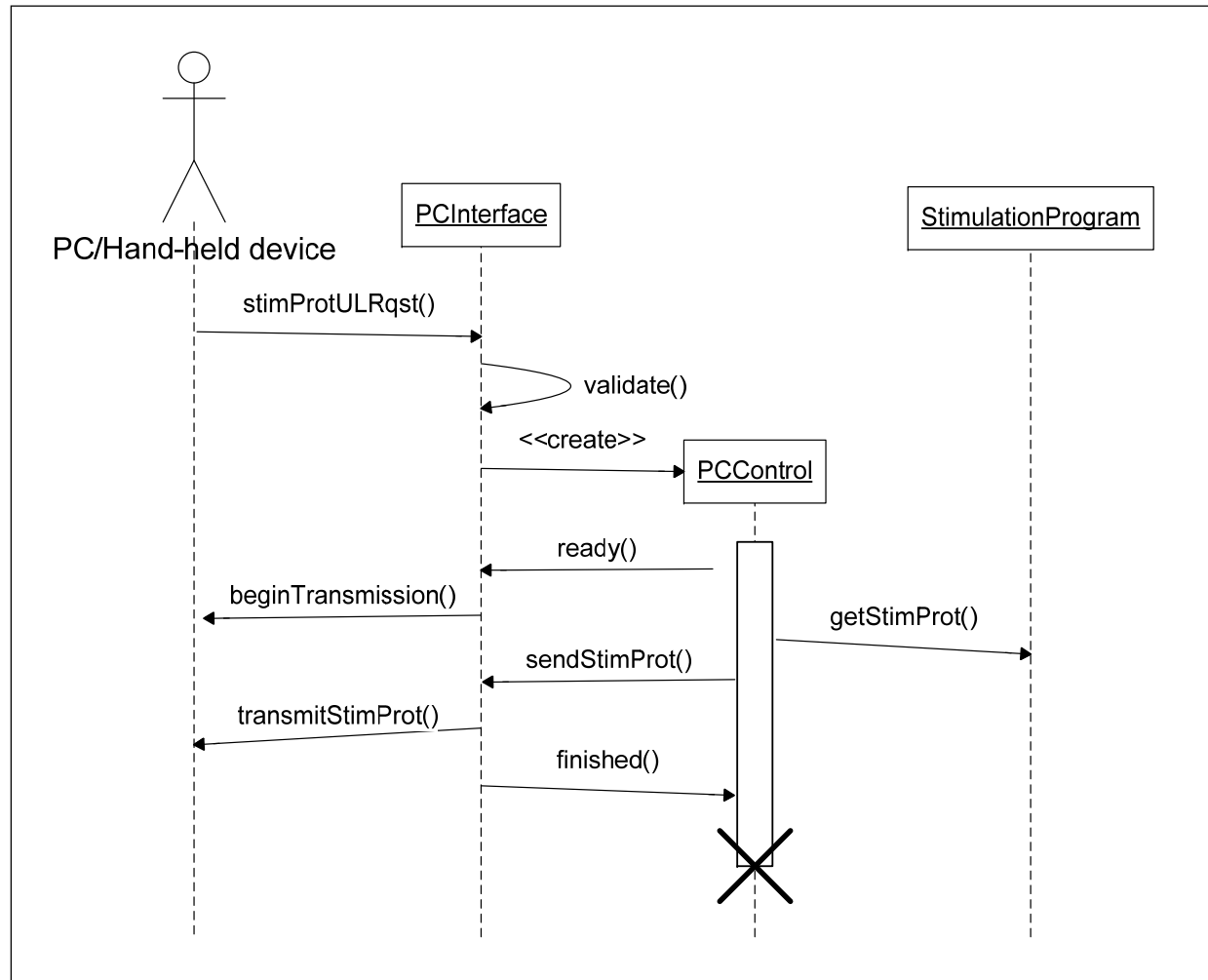


Figure A2-1: Static System Model – UML Class Diagram

## Appendix 3: Dynamic System Models

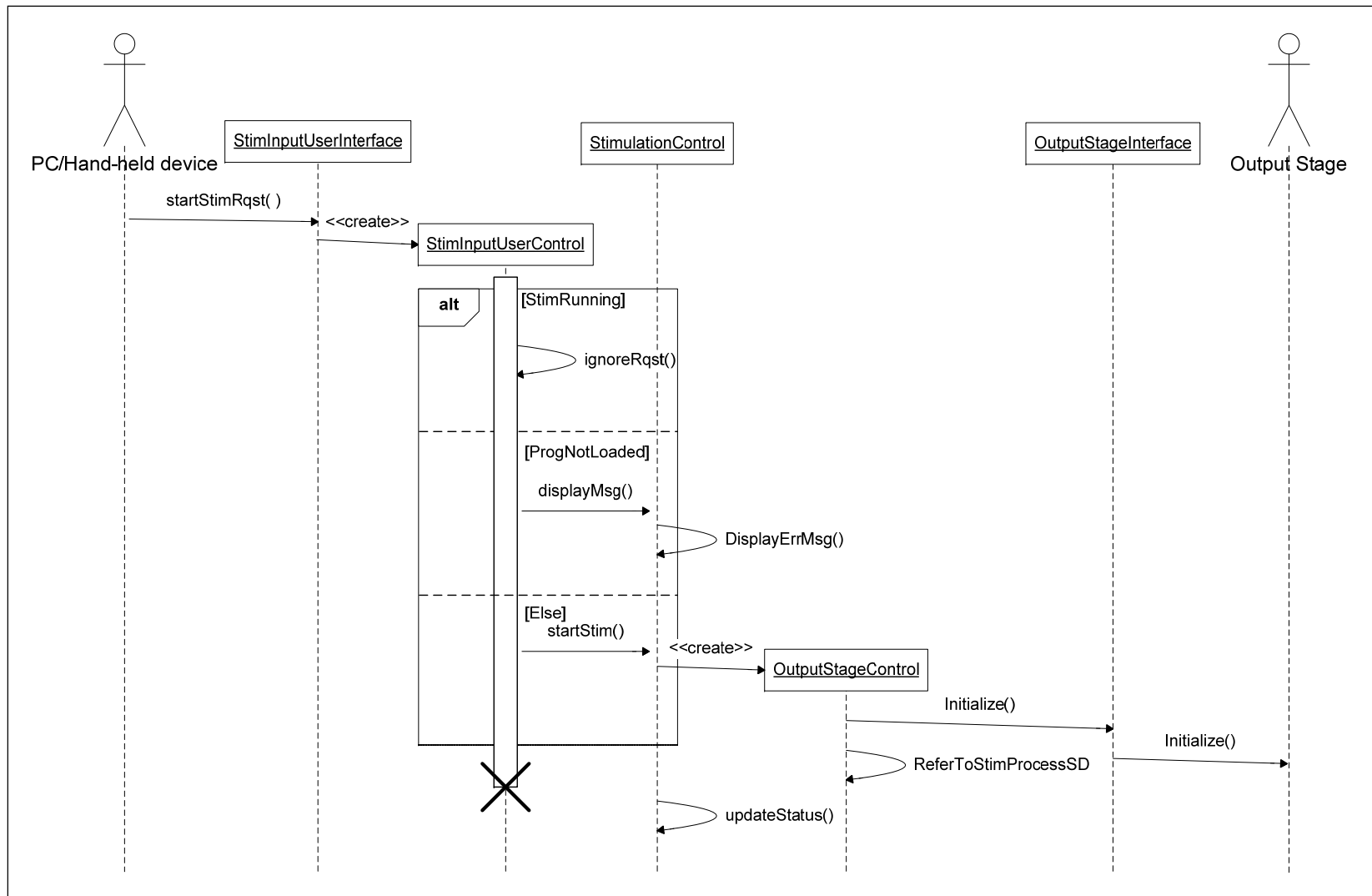


**Figure A3-1: Download Stimulation Protocol Sequence Diagram**

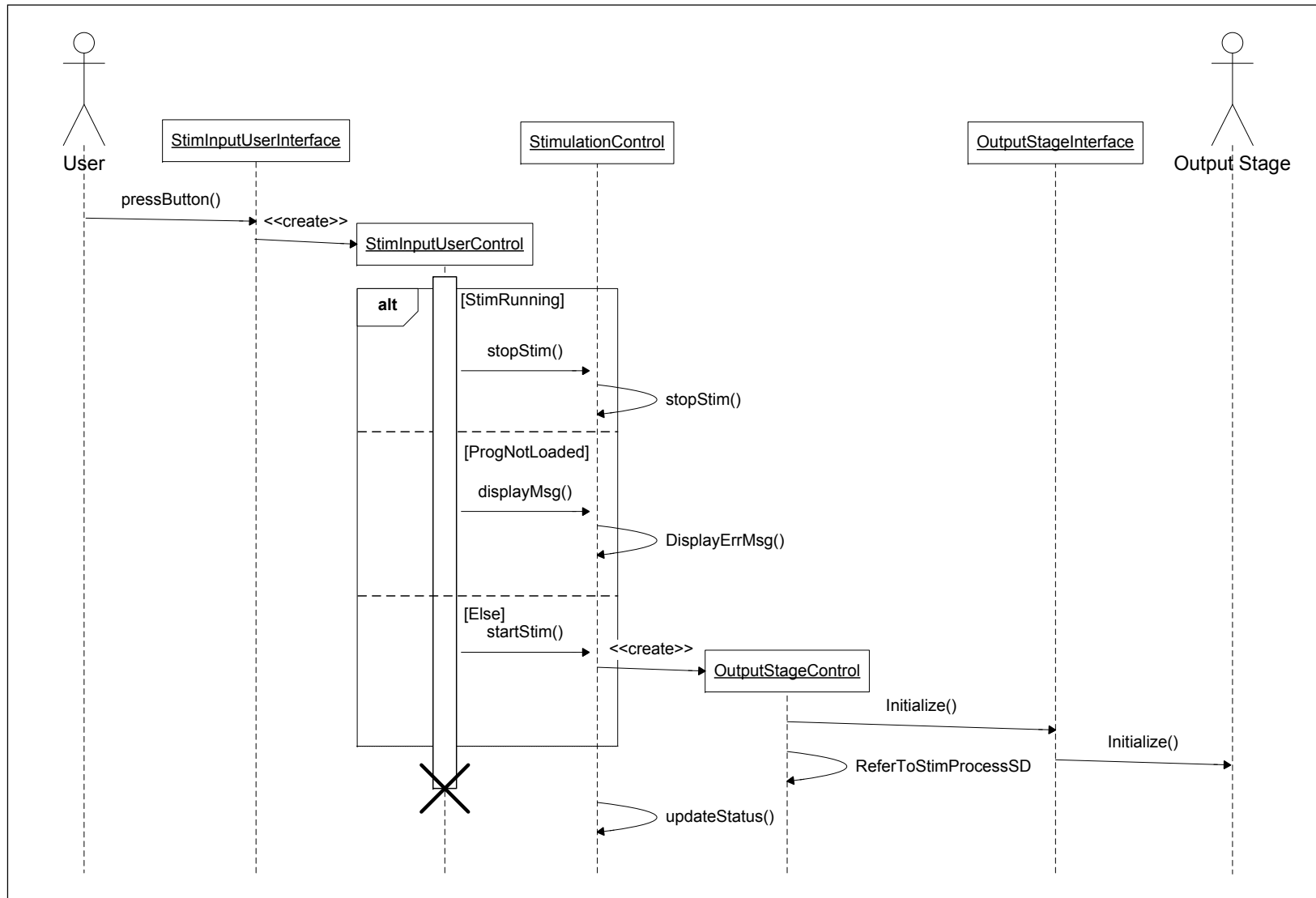


**Figure A3-2: Upload Stimulation Protocol Sequence Diagram**

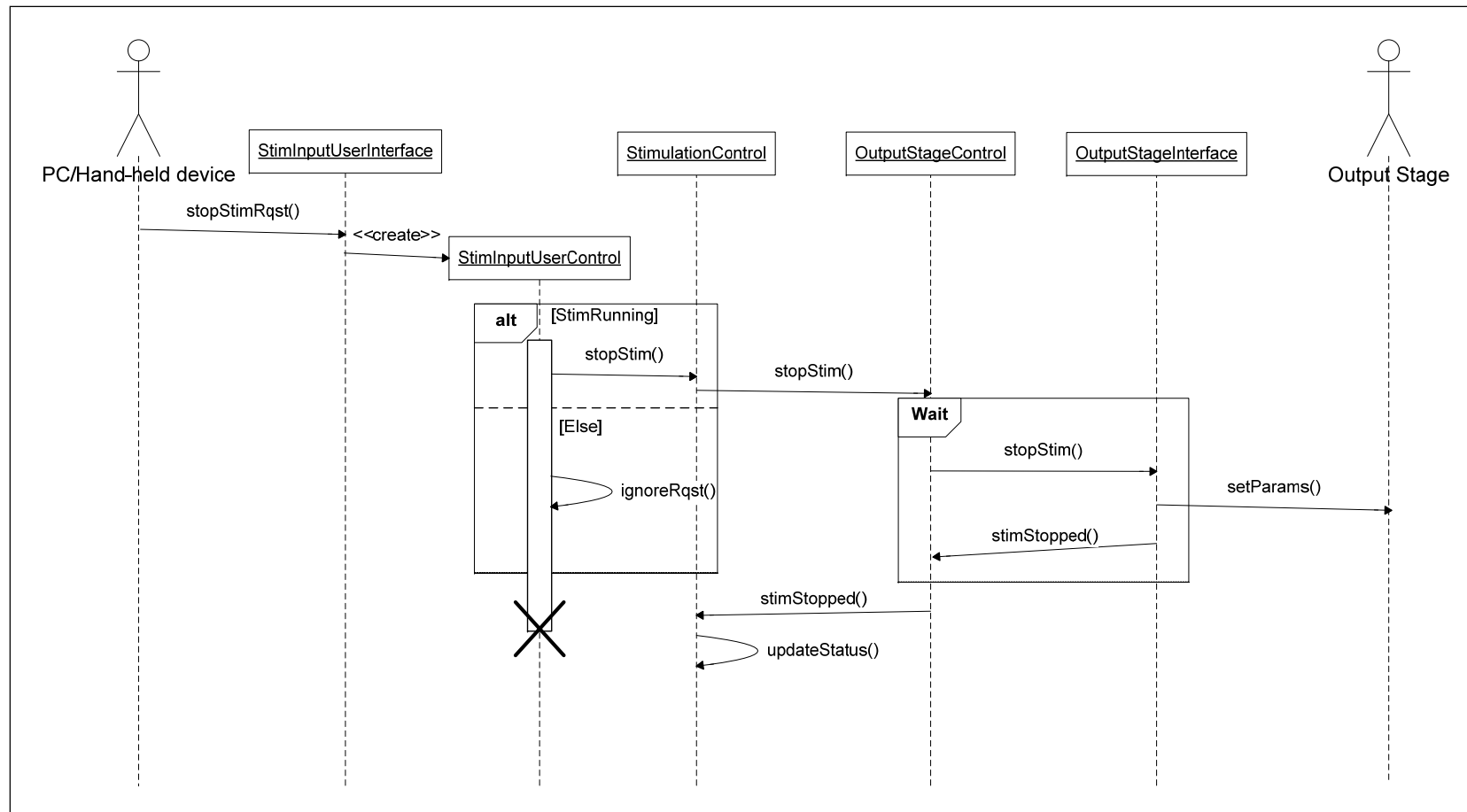




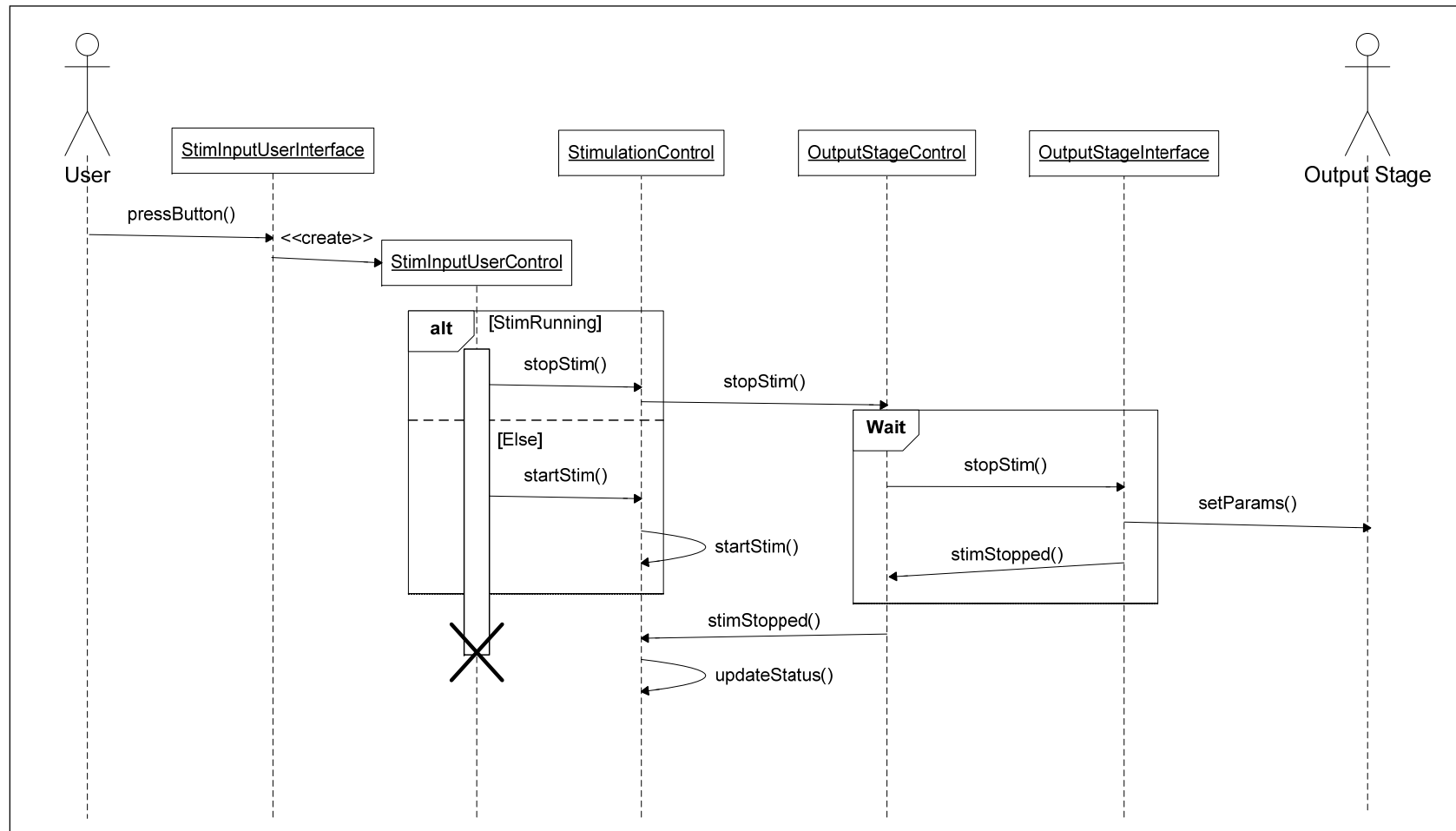
**Figure A3-3: Start Stimulation (PC/hand-held Device) Sequence Diagram**



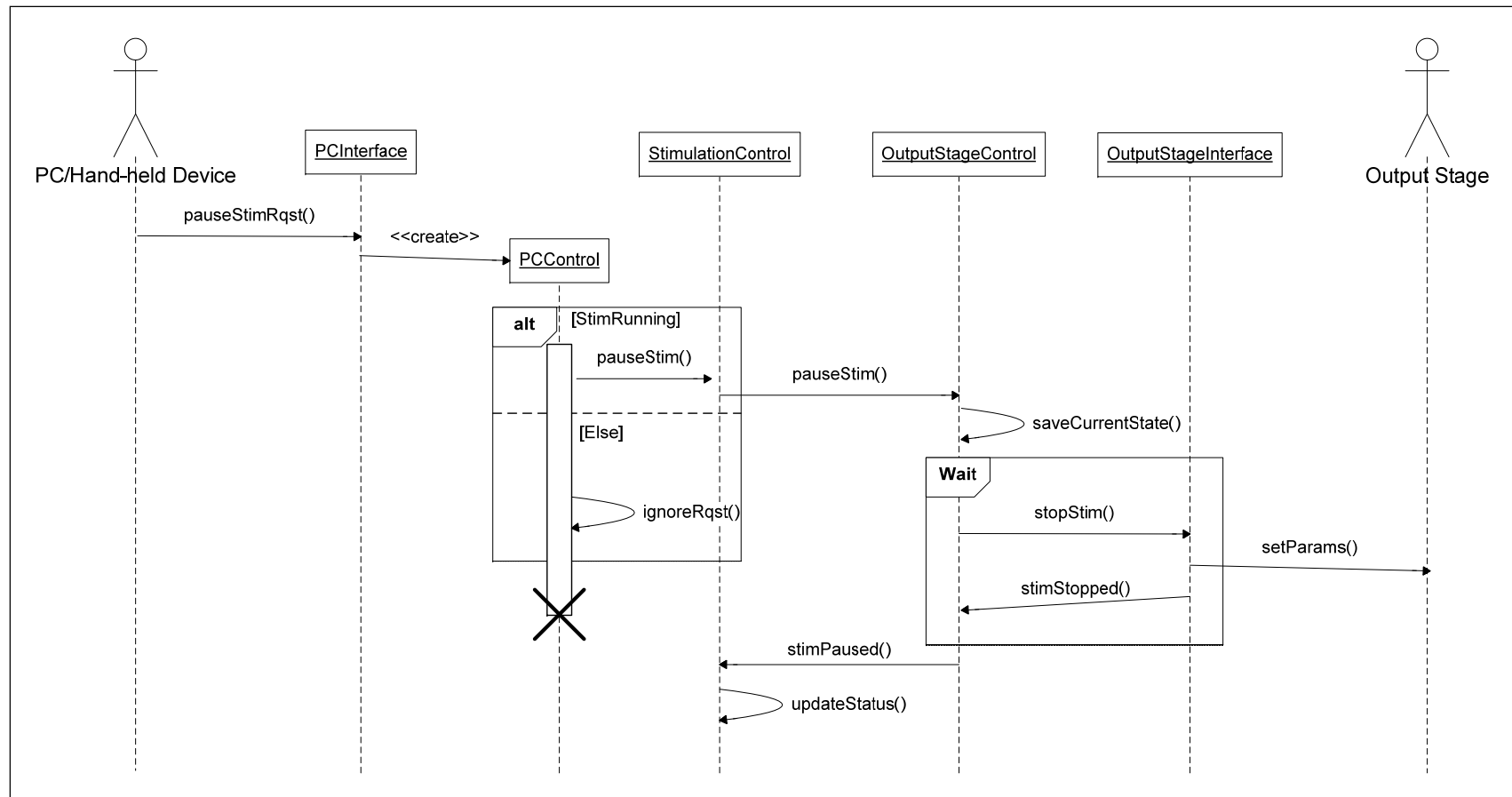
**Figure A3-4: Start Stimulation (user) Sequence Diagram**



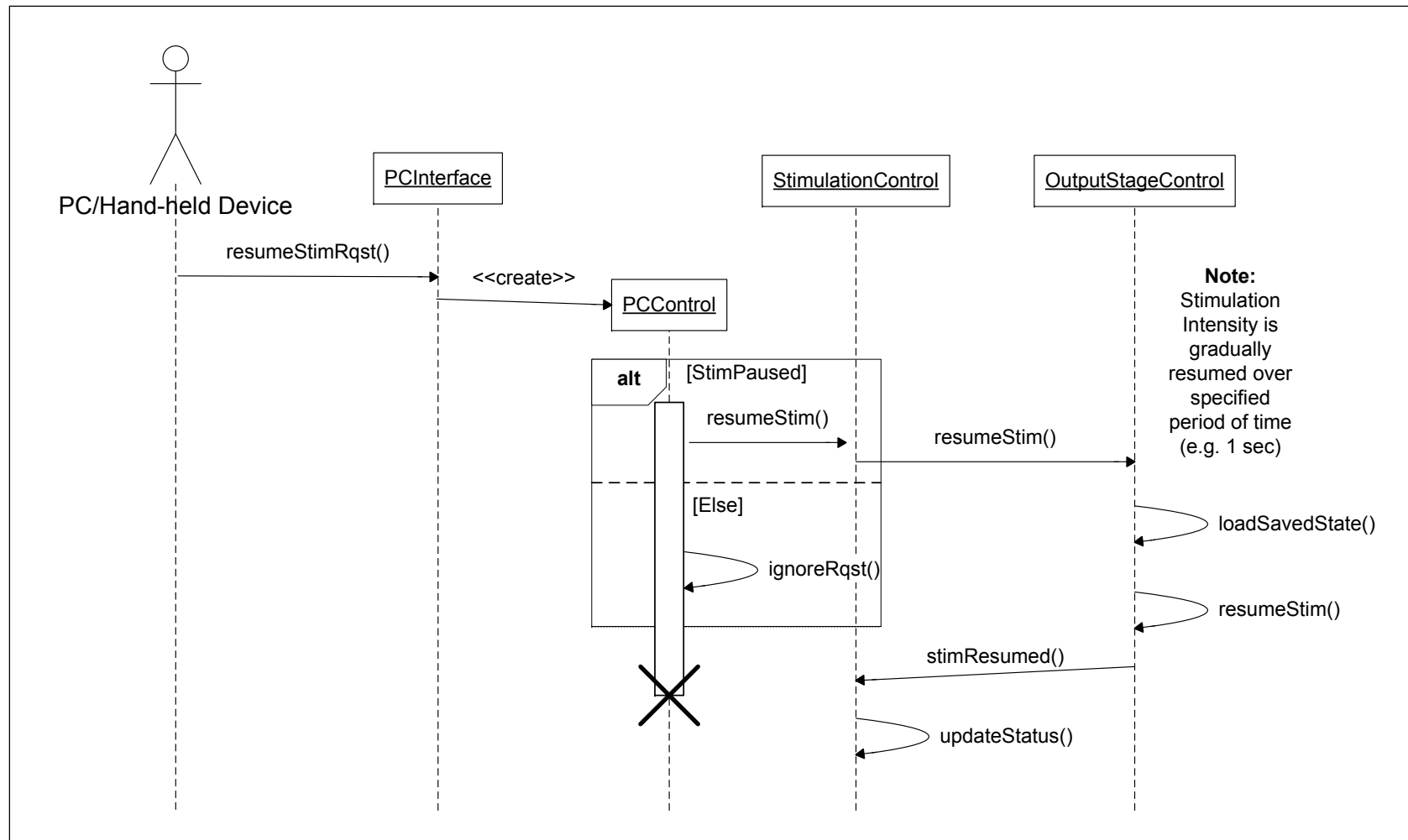
**Figure A3-5: Stop Stimulation (PC/hand-held Device) Sequence Diagram**



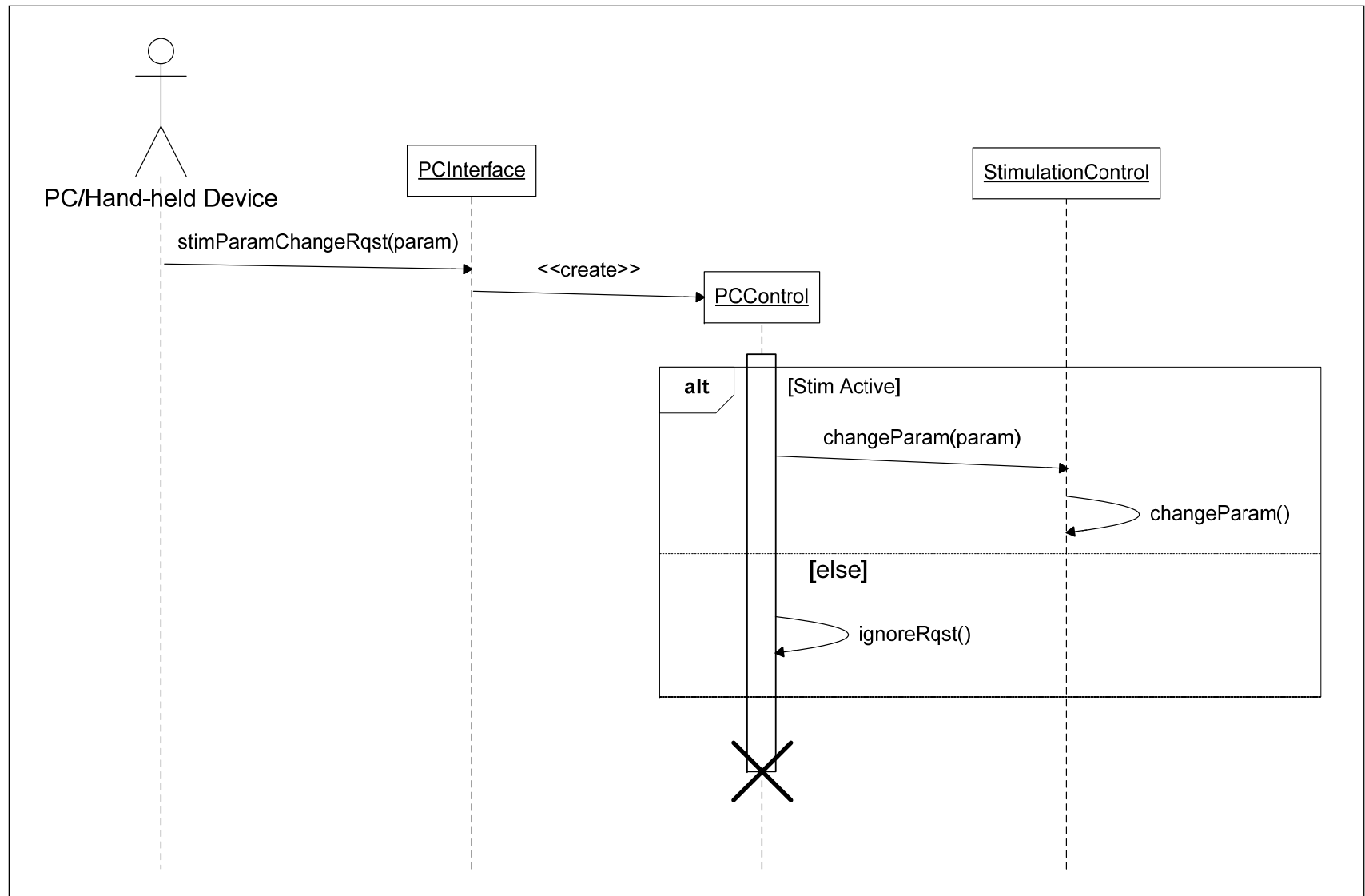
**Figure A3-6: Stop Stimulation (user) Sequence Diagram**



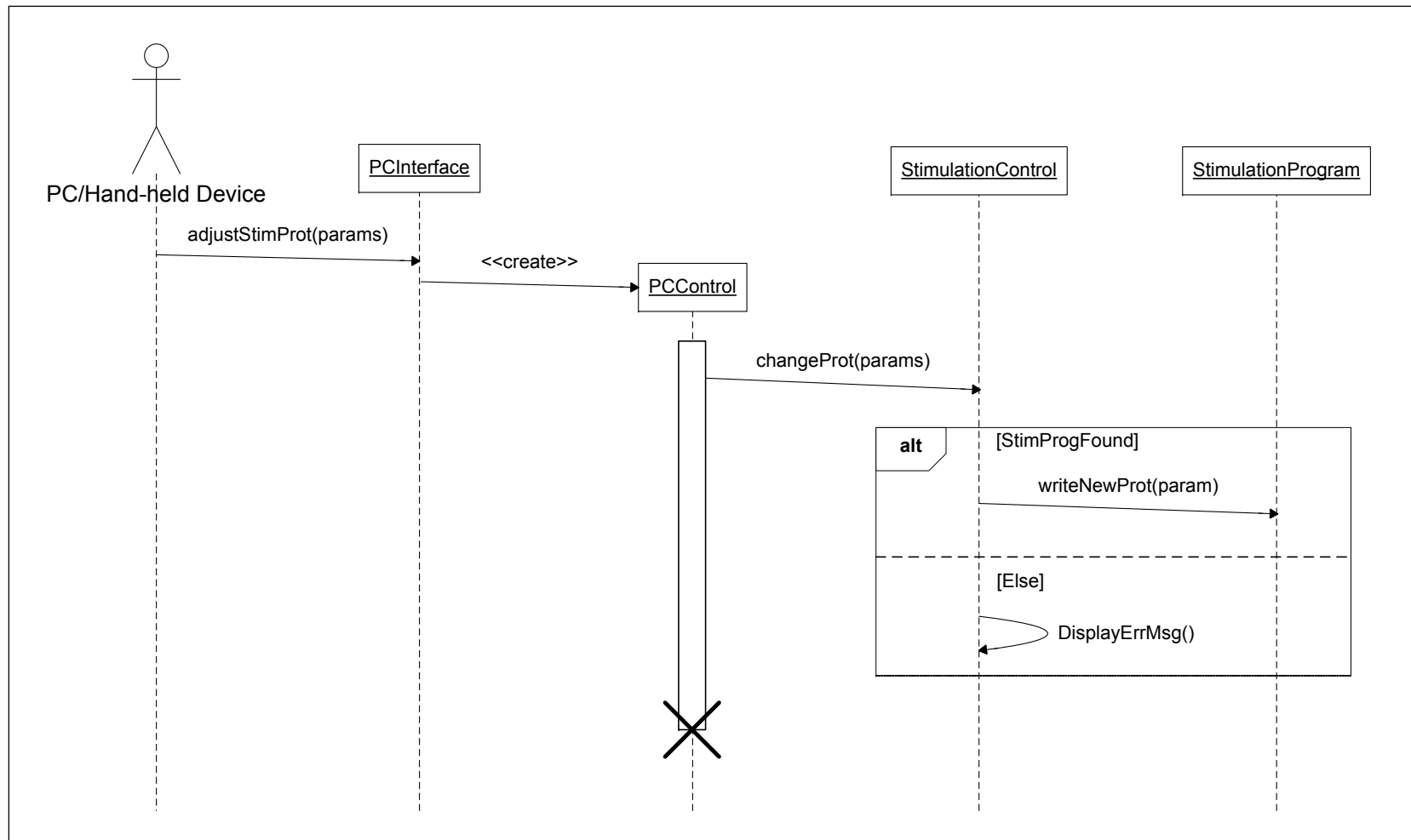
**Figure A3-7: Pause Stimulation Sequence Diagram**



**Figure A3-8: Resume Stimulation Sequence Diagram**

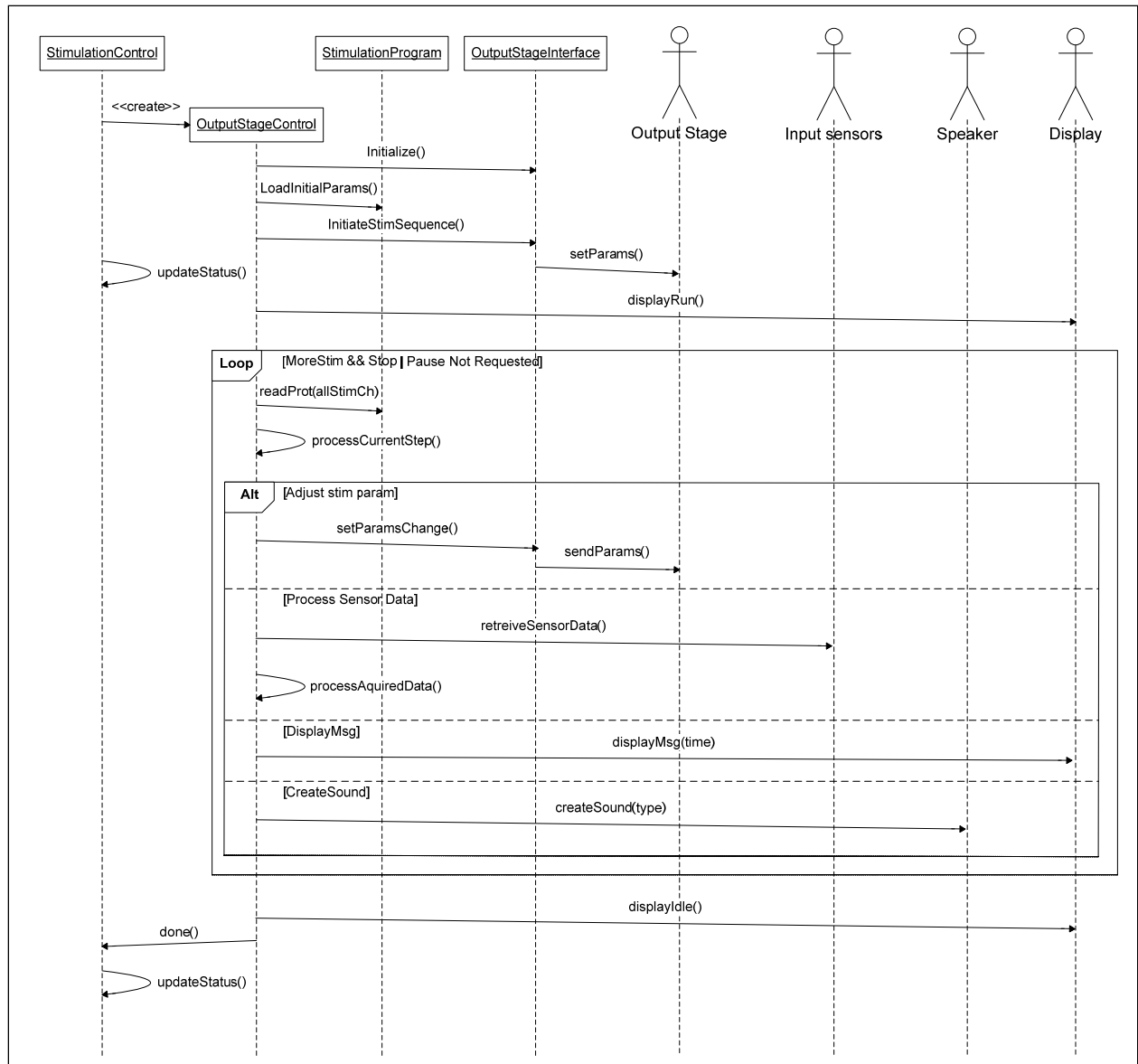


**Figure A3-9: Adjust Stimulation Parameters Sequence Diagram**



**Figure A3-10: Adjust Stimulation Protocol Sequence Diagram**





**Figure A3-11: Process Stimulation Protocol Sequence Diagram**

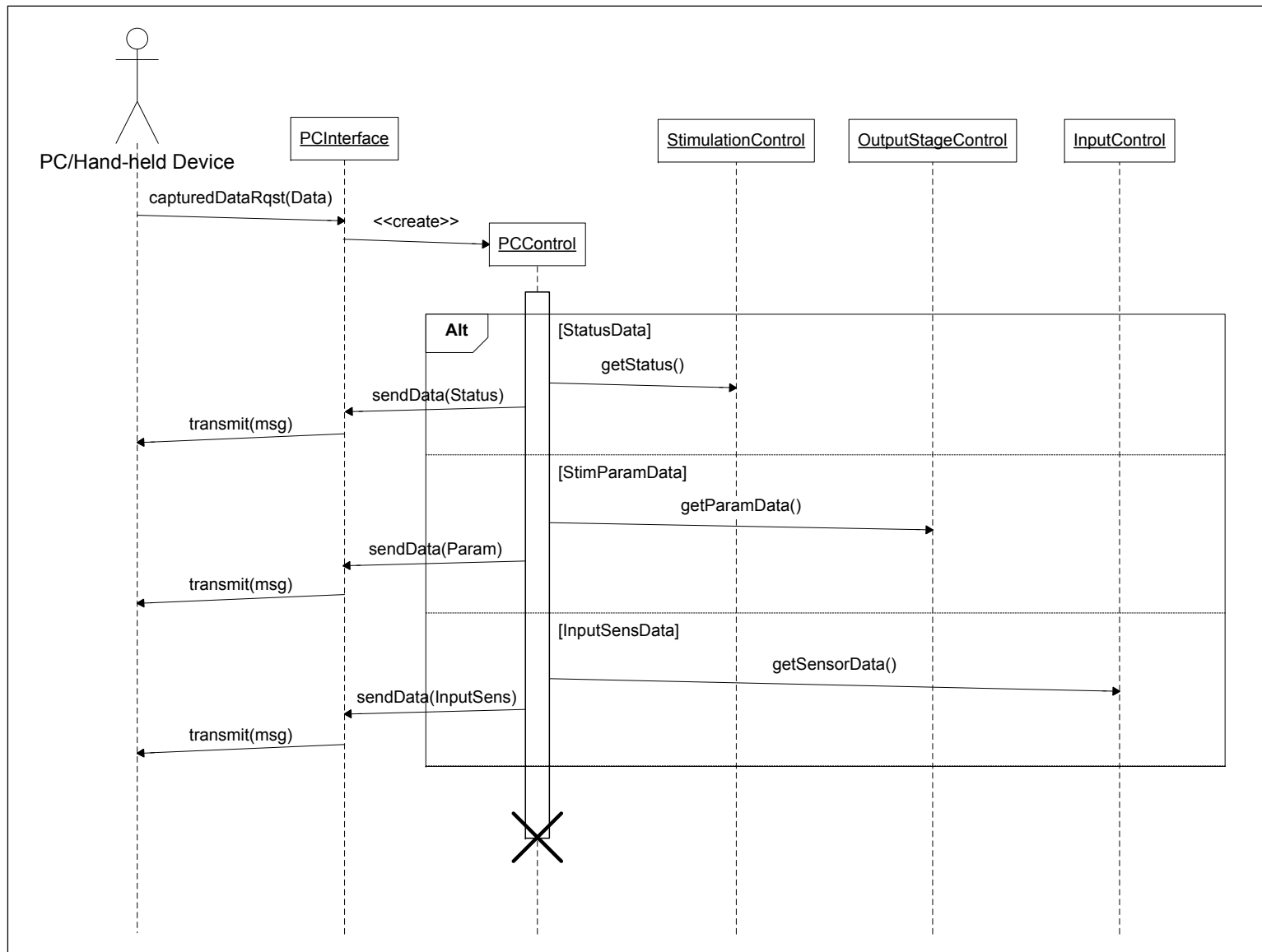
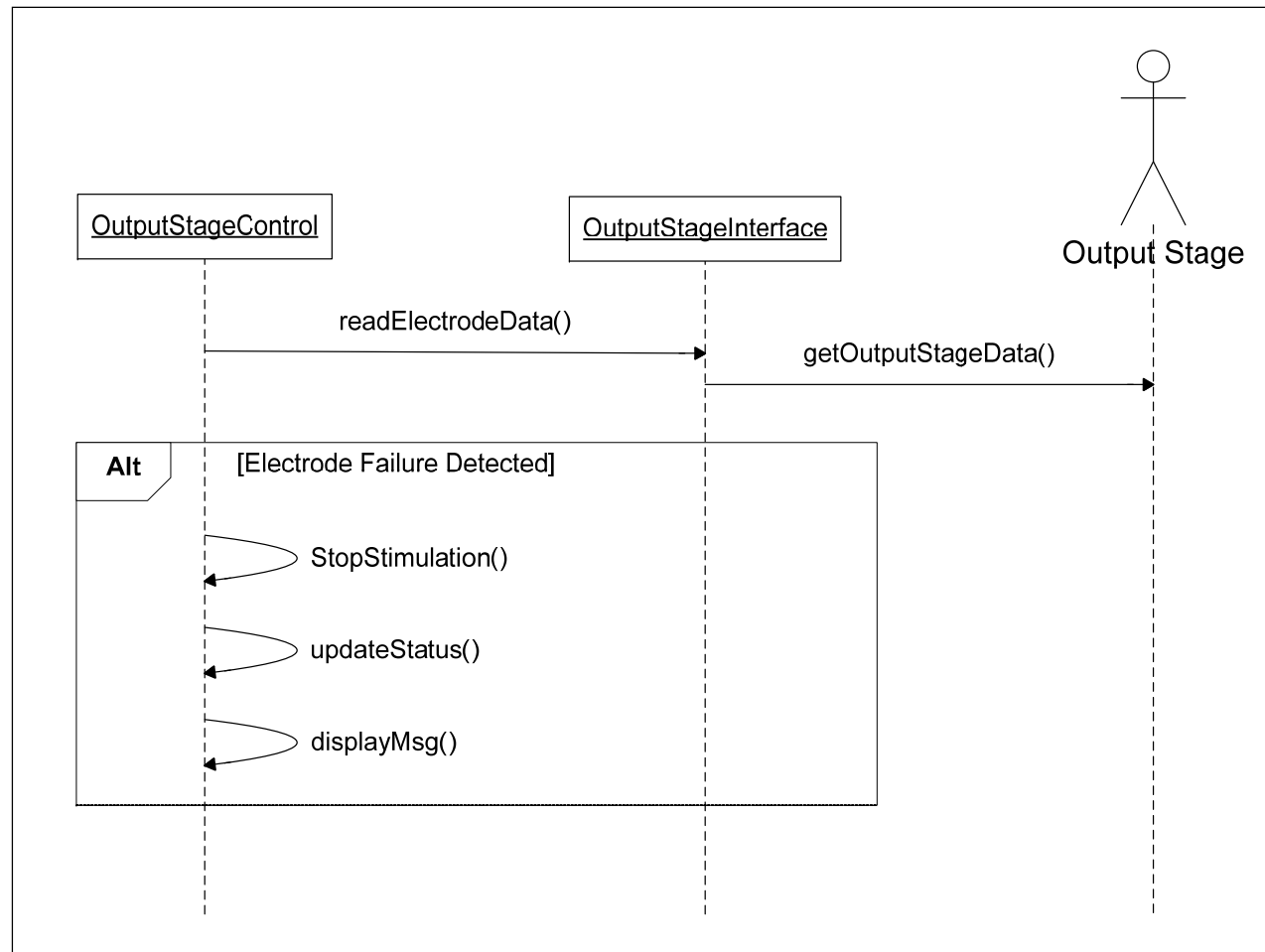
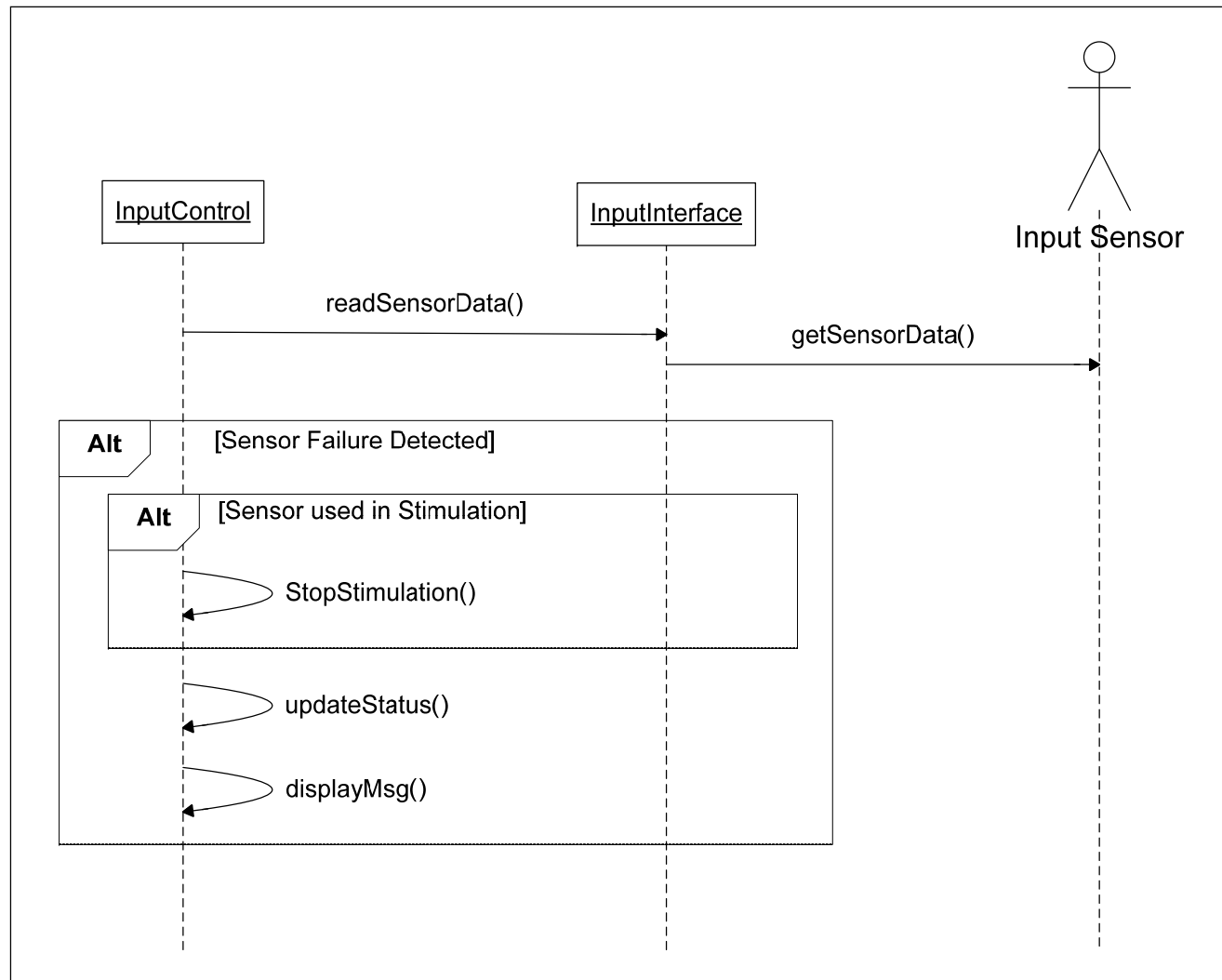


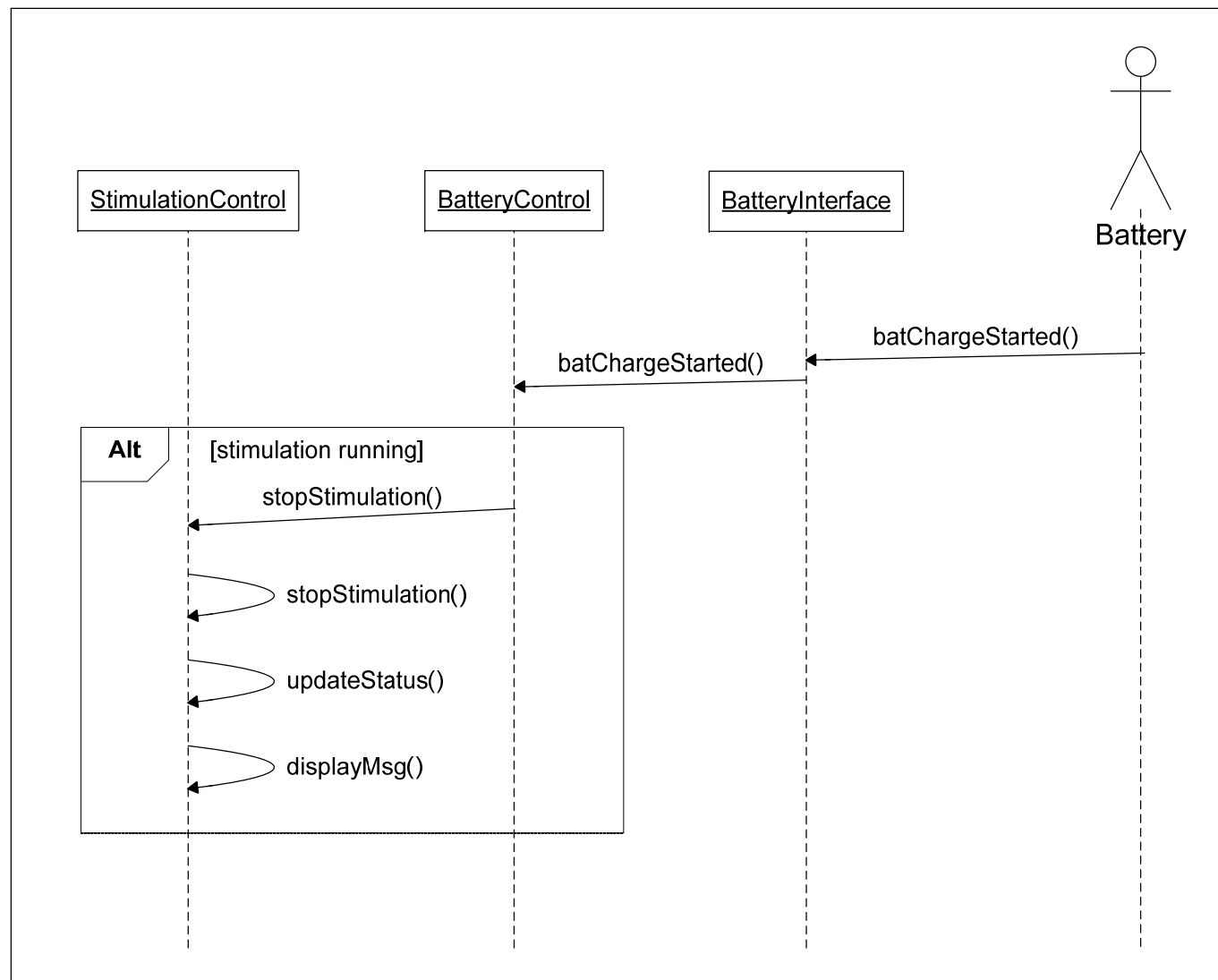
Figure A3-12: Capture Data Sequence Diagram



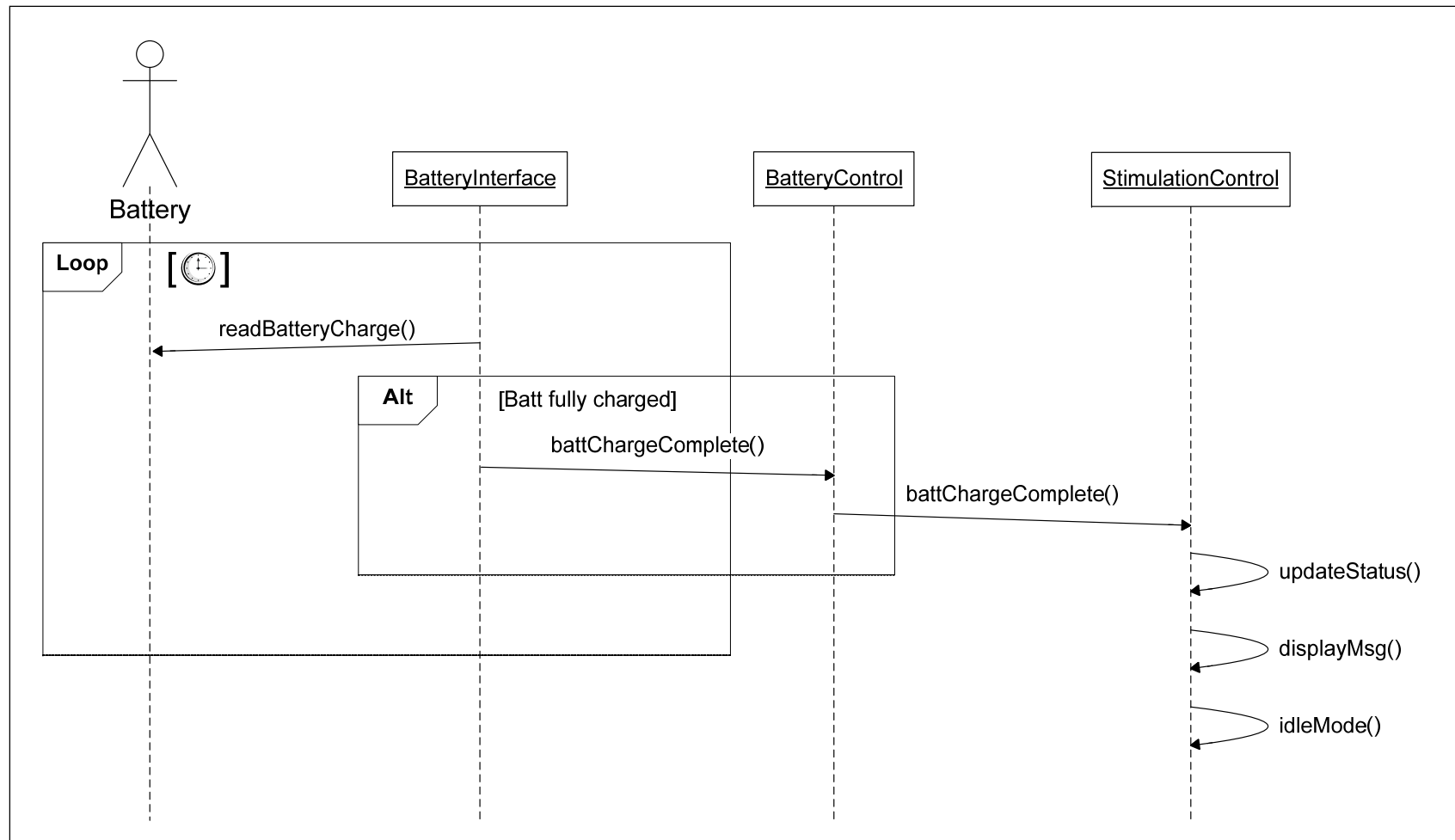
**Figure A3-13: Electrode Failure Sequence Diagram**



**Figure A3-14: Sensor Failure Sequence Diagram**



**Figure A3-15: Battery Charge in Progress Sequence Diagram**



**Figure A3-16: Battery Fully Charged Sequence Diagram**

## Appendix 4: Stimulation Protocol Encoding

Address (HEX)	Bytes	Comments
0000	1	Letter "F" indicating FES CARD I.D.
0001	1	Letter "B" indicating BIOFEEDBACK CARD I.D.
0002	1	Stimulator I.D. codes
0003	1	No. of currently used stimulator
0004	1	Month when the stimulator was manufactured
0005 -> 0010	12	12 characters that give the name to the card
0011	1	Always 00H
0012 -> 0016	5	5 characters card password
0017	1	Always 00H
0018 -> 001E	7	Not used
001F -> 0036	24	Setup filename
0037 -> 004C	22	Treatment
004D -> 0058	12	Subject's name
0059 -> 0064	12	Identification No.
0065 -> 006E	10	Date of Birth
006F -> 007A	12	Programmer's name
007B -> 007E	4	Date of last change
007F -> 0080	2	Minutes of use
0081 -> 0082	2	Minutes of stimulation
0083 -> 0089	7	Muscle name Ch1
008A -> 0090	7	Muscle name Ch2
0091 -> 0097	7	Muscle name Ch3
0098 -> 009E	7	Muscle name Ch4
009F -> 00AC	13	Not used
00AD	1	Timeout Constant (0 = infinite)
00AE	1	Gain A
00AF	1	Offset A
00B0	1	Gain B
00B1	1	Offset B
00B2	1	P VALUE A
00B3	1	I VALUE A
00B4	1	D VALUE A
00B5	1	P VALUE B
00B6	1	I VALUE B
00B7	1	D VALUE B
00B8	1	SCALEBIO A
00B9-00BA	2	ZEROBIO A
00BB	1	SCALEBIO B
00BC-00BD	2	ZEROBIO B

00BE	1	Analog AMP control input Ch1
00BF	1	Analog AMP control input Ch2
00C0	1	Analog AMP control input Ch3
00C1	1	Analog AMP control input Ch4
00C2	1	User interaction input & type 1
00C3	1	LVL1 - 1
00C4	1	T PH1 - 1
00C5	1	T PH2 - 1
00C6	1	LVL3 - 1
00C7	1	T PH3 - 1
00C8	1	User interaction input & type 2
00C9	1	LVL1 - 2
00CA	1	T PH1 - 2
00CB	1	T PH2 - 2
00CC	1	LVL3 - 2
00CD	1	T PH3 - 2
00CE	1	User interaction input & type 3
00CF	1	LVL1 - 3
00D0	1	T PH1 - 3
00D1	1	T PH2 - 3
00D2	1	LVL3 - 3
00D3	1	T PH3 - 3
00D4	1	User interaction input & type 4
00D5	1	LVL1 - 4
00D6	1	T PH1 - 4
00D7	1	T PH2 - 4
00D8	1	LVL3 - 4
00D9	1	T PH3 - 4
00DA	1	User interaction input & type 5
00DB	1	LVL1 - 5
00DC	1	T PH1 - 5
00DD	1	T PH2 - 5
00DE	1	LVL3 - 5
00DF	1	T PH3 - 5
00E0	1	User interaction input & type 6
00E1	1	LVL1 - 6
00E2	1	T PH1 - 6
00E3	1	T PH2 - 6
00E4	1	LVL3 - 6
00E5	1	T PH3 - 6
00E6	1	User interaction input & type 7



00E7	1	LVL1 - 7
00E8	1	T_PH1 - 7
00E9	1	T_PH2 - 7
00EA	1	LVL3 - 7
00EB	1	T_PH3 - 7
00EC	1	Branch 1 interaction continue
00ED	1	Branch 1 interaction jump
00EE	1	Branch 2 interaction continue
00EF	1	Branch 2 interaction jump
00F0	1	User interrupt
00F1	1	Branch 1 jump address Ch1
00F2	1	Branch 2 jump address Ch1
00F3	1	User interrupt jump address Ch1
00F4	1	Branch 1 jump address Ch2
00F5	1	Branch 2 jump address Ch2
00F6	1	User interrupt jump address Ch2
00F7	1	Branch 1 jump address Ch3
00F8	1	Branch 2 jump address Ch3
00F9	1	User interrupt jump address Ch3
00FA	1	Branch 1 jump address Ch4
00FB	1	Branch 2 jump address Ch4
00FC	1	User interrupt jump address Ch4
00FD -> 00FE	2	Time to adjust new amplitude 1 Ch1
00FF -> 0100	2	Time to adjust new amplitude 2 Ch1
0101 -> 0102	2	Time to adjust new amplitude 3 Ch1
0103 -> 0104	2	Time to adjust new amplitude 4 Ch1
0105 -> 0106	2	Time to adjust new amplitude 1 Ch2
0107 -> 0108	2	Time to adjust new amplitude 2 Ch2
0109 -> 010A	2	Time to adjust new amplitude 3 Ch2
010B -> 010C	2	Time to adjust new amplitude 4 Ch2
010D -> 010E	2	Time to adjust new amplitude 1 Ch3
010F -> 0110	2	Time to adjust new amplitude 2 Ch3
0111 -> 0112	2	Time to adjust new amplitude 3 Ch3
0113 -> 0114	2	Time to adjust new amplitude 4 Ch3
0115 -> 0116	2	Time to adjust new amplitude 1 Ch4
0117 -> 0118	2	Time to adjust new amplitude 2 Ch4
0119 -> 011A	2	Time to adjust new amplitude 3 Ch4
011B -> 011C	2	Time to adjust new amplitude 4 Ch4
011D -> 011E	2	Check sum

011F	1	Ch1 maximum pulse amplitude
0120	1	Ch2 maximum pulse amplitude
0121	1	Ch3 maximum pulse amplitude
0122	1	Ch4 maximum pulse amplitude
0123	1	Ch1 default pulse amplitude
0124	1	Ch2 default pulse amplitude
0125	1	Ch3 default pulse amplitude
0126	1	Ch4 default pulse amplitude
0127 -> 0128	2	Ch1 maximum pulse width
0129 -> 012A	2	Ch2 maximum pulse width
012B -> 012C	2	Ch3 maximum pulse width
012D -> 012E	2	Ch4 maximum pulse width
012F -> 0130	2	Ch1 constant No.1 - pulse width
0131 -> 0132	2	Ch1 constant No.2 - pulse width
0133 -> 0134	2	Ch1 constant No.3 - pulse width
0135 -> 0136	2	Ch1 constant No.4 - pulse width
0137 -> 0138	2	Ch2 constant No.1 - pulse width
0139 -> 013A	2	Ch2 constant No.2 - pulse width
013B -> 013C	2	Ch2 constant No.3 - pulse width
013D -> 013E	2	Ch2 constant No.4 - pulse width
013F -> 0140	2	Ch3 constant No.1 - pulse width
0141 -> 0142	2	Ch3 constant No.2 - pulse width
0143 -> 0144	2	Ch3 constant No.3 - pulse width
0145 -> 0146	2	Ch3 constant No.4 - pulse width
0147 -> 0148	2	Ch4 constant No.1 - pulse width
0149 -> 014A	2	Ch4 constant No.2 - pulse width
014B -> 014C	2	Ch4 constant No.3 - pulse width
014D -> 014E	2	Ch4 constant No.4 - pulse width
014F -> 0150	2	Stimulation frequency
0151	1	Amplitude control
0152	1	Stimulation mode
0153	1	Text1 display time
0154	1	Text2 display time
0155	1	Text3 display time
0156	1	Text4 display time
0157	1	New amplitude 1 Ch1
0158	1	New amplitude 2 Ch1
0159	1	New amplitude 3 Ch1
015A	1	New amplitude 4 Ch1
015B	1	New amplitude 1 Ch2
015C	1	New amplitude 2 Ch2

015D	1	New amplitude 3 Ch2
015E	1	New amplitude 4 Ch2
015F	1	New amplitude 1 Ch3
0160	1	New amplitude 2 Ch3
0161	1	New amplitude 3 Ch3
0162	1	New amplitude 4 Ch3
0163	1	New amplitude 1 Ch4
0164	1	New amplitude 2 Ch4
0165	1	New amplitude 3 Ch4
0166	1	New amplitude 4 Ch4
0167 -> 0168	2	New frequency 1
0169 -> 016A	2	New frequency 2
016B -> 016C	2	New frequency 3
016D -> 016E	2	New frequency 4
016F	1	Random amplitude Ch1
0170	1	Random amplitude Ch2
0171	1	Random amplitude Ch3
0172	1	Random amplitude Ch4
0173	1	Random frequency
0174	1	Random pulse width Ch1
0175	1	Random pulse width Ch2
0176	1	Random pulse width Ch3
0177	1	Random pulse width Ch4
0178 -> 0179	2	Not used
017A -> 0199	32	Text for the text marker No.1
019A -> 01B9	32	Text for the text marker No.2
01BA -> 01D9	32	Text for the text marker No.3
01DA -> 01F9	32	Text for the text marker No.4
01FA -> 01FB	2	Check sum
01FC -> 023B	64	Ch1 amplitude look-up table
023C -> 027B	64	Ch2 amplitude look-up table
027C -> 02BB	64	Ch3 amplitude look-up table
02BC -> 02FB	64	Ch4 amplitude look-up table
02FC -> 02FD	2	Check sum
02FE -> 031D	32	Ch1 - ramp 1 (2 bytes * 16 values)
031E -> 033D	32	Ch2 - ramp 1 (2 bytes * 16 values)
033E -> 035D	32	Ch3 - ramp 1 (2 bytes * 16 values)
035E -> 037D	32	Ch4 - ramp 1 (2 bytes * 16 values)
037E -> 039D	32	Ch1 - ramp 2 (2 bytes * 16 values)
039E -> 03BD	32	Ch2 - ramp 2 (2 bytes * 16 values)
03BE -> 03DD	32	Ch3 - ramp 2 (2 bytes * 16 values)

03DE -> 03FD	32	Ch4 - ramp 2 (2 bytes * 16 values)
03FE -> 03FF	2	Check sum
0400 -> 04FD	254	Ch1 string of primitives; 1B each; last is 255
04FE -> 04FF	2	Check sum
0500 -> 05FD	254	Ch2 string of primitives; 1B each; last is 255
05FE -> 05FF	2	Check sum
0600 -> 06FD	254	Ch3 string of primitives; 1B each; last is 255
06FE -> 06FF	2	Check sum
0700 -> 07FD	254	Ch4 string of primitives; 1B each; last is 255
07FE -> 07FF	2	Check sum

**Table A4-1: Stimulation protocol byte encoding details.**

Primitive & Number	Function	Parameters	Address (HEX)
Ramp A up (0-31)	Takes Ramp A up and calculates ramp depending on value: 0 -> 31 equivalent to: 1 cycle -> 32 cycles (step: 1 cycle)	Ch1: Ramp A (2B) Ch2: Ramp A (2B) Ch3: Ramp A (2B) Ch4: Ramp A (2B)	02FE -> 031D 031E -> 033D 033E -> 035D 035E -> 037D
Ramp A down (32-63)	Takes Ramp A down and calculates ramp depending on value: 32 -> 63 equivalent to: 1 cycle -> 32 cycles (step: 1 cycle)	Ch1: Ramp A (2B) Ch2: Ramp A (2B) Ch3: Ramp A (2B) Ch4: Ramp A (2B)	02FE -> 031D 031E -> 033D 033E -> 035D 035E -> 037D
Ramp B up (64-95)	Takes Ramp B up and calculates ramp depending on value: 64 -> 95 equivalent to: 1 cycle -> 32 cycles (step 1: cycle)	Ch1: Ramp B (2B) Ch2: Ramp B (2B) Ch3: Ramp B (2B) Ch4: Ramp B (2B)	037E -> 039D 039E -> 03BD 03BE -> 03DD 03DE -> 03FD
Ramp B down (96-127)	Takes Ramp B down and calculates ramp depending on value: 96 -> 127 equivalent to: 1 cycle -> 32 cycles (step 1: cycle)	Ch1: Ramp B (2B) Ch2: Ramp B (2B) Ch3: Ramp B (2B) Ch4: Ramp B (2B)	037E -> 039D 039E -> 03BD 03BE -> 03DD 03DE -> 03FD
Constant A pulse width (128-135)	Takes PW A and calculates number of cycles to keep const value: 128->135 128:1cycle, 129:2 cycle, 130:4cycle, 131:8cycle, 132:16 cycle, 133:32 cycle, 134:64 cycle, 135:128 cycle.	Ch1: Const PW A (2B) Ch2: Const PW A (2B) Ch3: Const PW A (2B) Ch4: Const PW A (2B)	012F -> 0130 0137 -> 0138 013F -> 0140 0147 -> 0148
Constant B pulse width (136-143)	Takes PW B and calculates number of cycles to keep const value: 136->146 Same pattern as above.	Ch1: Const PW B (2B) Ch2: Const PW B (2B) Ch3: Const PW B (2B) Ch4: Const PW B (2B)	0131 -> 0132 0139 -> 013A 0141 -> 0142 0149 -> 014A
Constant C pulse width (144-151)	Takes PW C and calculates number of cycles to keep const value: 144->151 Same pattern as above.	Ch1: Const PW C (2B) Ch2: Const PW C (2B) Ch3: Const PW C (2B) Ch4: Const PW C (2B)	0133 -> 0134 013B -> 013C 0143 -> 0144 0143 -> 0144
Constant D pulse width (152-159)	Takes PW D and calculates number of cycles to keep const value: 152->159 Same pattern as above.	Ch1: Const PW D (2B) Ch2: Const PW D (2B) Ch3: Const PW D (2B) Ch4: Const PW D (2B)	0135 -> 0136 013D -> 013E 0145 -> 0146 014D -> 014E

No Stimulation (160-167)	Takes PW of 0 and calculates numberof cycles to keep 0 PW: 160->167160:1 cycle,161:2 cycle,162:4 cycle,163:8 cycle,164:16 cycle,165:32cycle,166:64 cycles,167:128 cycles	N/A N/A N/A N/A N/A	N/A N/A N/A N/A N/A
Marker A (168)	Index location for Marker A (M-A)	N/A	N/A
Jump A (169)	Loop once to M-A	N/A	N/A
Jump A (170)	Loop twice to M-A	N/A	N/A
Jump A (171)	Loop four times to M-A	N/A	N/A
Jump A (172)	Loop eight times to M-A	N/A	N/A
Jump A (173)	Loop 16 times to M-A	N/A	N/A
Jump A (174)	Loop 32 times to M-A	N/A	N/A
Jump A (175)	Loop 64 times to M-A	N/A	N/A
Jump A (176)	Loop 128 times to M-A	N/A	N/A
Marker B (177)	Index location for Marker B (M-B)	N/A	N/A
Jump B (178)	Loop once to M-B	N/A	N/A
Jump B (179)	Loop twice to M-B	N/A	N/A
Jump B (180)	Loop four times to M-B	N/A	N/A
Jump B (181)	Loop eight times to M-B	N/A	N/A
Jump B (182)	Loop 16 times to M-B	N/A	N/A
Jump B (183)	Loop 32 times to M-B	N/A	N/A
Jump B (184)	Loop 64 times to M-B	N/A	N/A
Jump B (185)	Loop 128 times to M-B	N/A	N/A
Marker C (186)	Index location for Marker C (M-C)	N/A	N/A
Jump C (187)	Loop once to M-C	N/A	N/A
Jump C (188)	Loop twice to M-C	N/A	N/A
Jump C (189)	Loop four times to M-C	N/A	N/A
Jump C (190)	Loop eight times to M-C	N/A	N/A
Jump C (191)	Loop 16 times to M-C	N/A	N/A
Jump C (192)	Loop 32 times to M-C	N/A	N/A
Jump C (193)	Loop 64 times to M-C	N/A	N/A
Jump C (194)	Loop 128 times to M-C	N/A	N/A

Marker D (195)	Index location for Marker D (M-D)	N/A	N/A
Jump D (196)	Loop once to M-D	N/A	N/A
Jump D (197)	Loop twice to M-D	N/A	N/A
Jump D (198)	Loop four times to M-D	N/A	N/A
Jump D (199)	Loop eight times to M-D	N/A	N/A
Jump D (200)	Loop 16 times to M-D	N/A	N/A
Jump D (201)	Loop 32 times to M-D	N/A	N/A
Jump D (202)	Loop 64 times to M-D	N/A	N/A
Jump D (203)	Loop 128 times to M-D	N/A	N/A
Delay (204)	Delay 1 cycle	N/A	N/A
Delay (205)	Delay 2 cycles	N/A	N/A
Delay (206)	Delay 4 cycles	N/A	N/A
Delay (207)	Delay 8 cycles	N/A	N/A
Delay (208)	Delay 16 cycles	N/A	N/A
Delay (209)	Delay 32 cycles	N/A	N/A
Delay (210)	Delay 64 cycles	N/A	N/A
Delay (211)	Delay 128 cycles	N/A	N/A
Jump A (212)	Jump A infinite	N/A	N/A
Jump B (213)	Jump B infinite	N/A	N/A
Jump C (214)	Jump C infinite	N/A	N/A
Jump D (215)	Jump D infinite	N/A	N/A
Amp Change A (216)	Ch1: change Amplitude to Amp A Ch2: change Amplitude to Amp A Ch3: change Amplitude to Amp A Ch4: change Amplitude to Amp A	Ch1: Amp A(1B),time(2B) Ch2: Amp A(1B),time(2B) Ch3: Amp A(1B),time(2B) Ch4: Amp A(1B),time(2B)	0157, 00FD -> 00FE 015B, 0105 -> 0106 015F, 010D -> 010E 0163, 0115 -> 0116
Amp Change B (217)	Ch1: change Amplitude to Amp B Ch2: change Amplitude to Amp B Ch3: change Amplitude to Amp B Ch4: change Amplitude to Amp B	Ch1: Amp B(1B),time(2B) Ch2: Amp B(1B),time(2B) Ch3: Amp B(1B),time(2B) Ch4: Amp B(1B),time(2B)	0158, 00FF -> 0100 015C, 0107 -> 0108 0160, 010F -> 0110 0164, 0117 -> 0118
Amp Change C (218)	Ch1: change Amplitude to Amp C Ch2: change Amplitude to Amp C Ch3: change Amplitude to Amp C Ch4: change Amplitude to Amp C	Ch1: Amp C(1B),time(2B) Ch2: Amp C(1B),time(2B) Ch3: Amp C(1B),time(2B) Ch4: Amp C(1B),time(2B)	0159, 0101 -> 0102 015D, 0109 -> 010A 0161, 0111 -> 0112 0165, 0119 -> 011A

Amp Change D (219)	Ch1: change Amplitude to Amp D Ch2: change Amplitude to Amp D Ch3: change Amplitude to Amp D Ch4: change Amplitude to Amp D	Ch1: Amp D(1B),time(2B) Ch2: Amp D(1B),time(2B) Ch3: Amp D(1B),time(2B) Ch4: Amp D(1B),time(2B)	015A, 0103 -> 0104 015E, 010B -> 010C 0162, 0113 -> 0114 0166, 011B -> 011C
Frq Change A (220) Frq Change B (221) Frq Change C (222) Frq Change D (223)	Change frequency to Freq A Change frequency to Freq B Change frequency to Freq B Change frequency to Freq B	Freq A (2B) Freq B (2B) Freq C (2B) Freq D (2B)	0167 -> 0168 0169 -> 016A 016B -> 016C 016D -> 016E
Rand frq on (224)	Start random frequency	Rand % (1B)	0173
Rand frq off (225)	Stop random frequency	N/A	N/A
Rand PW on (226)	Start random pulse widths	Rand % (1B,ch1-4)	0174,0175,0176,0177
Rand PW off (227)	Stop random pulse widths	N/A	N/A
User branch A (228)	User branch based on user interaction.	Branch A continue (1B) Branch A jump (1B) Jump address (1B,ch1-4)	00EC 00ED 00F1,00F4,00F7,00FA
User branch B (229)	User branch based on user interaction.	Branch B continue (1B) Branch B jump (1B) Jump address (1B,ch1-4)	00EE 00EF 00F2,00F5,00F8,00FB
Turn off (230)	Turns off stimulator	N/A	N/A
User Int on (231)	User Interrupts on. User interaction to trigger interrupt.	User int interaction(1B) Int routine jump addrs (1B,ch1-ch4)	00F0 00F3,00F6,00F9,00FC
User Int off (232)	User Interrupts off	N/A	N/A
Text A (233) Text B (234) Text C (235) Text D (236)	Display text A Display text B Display text C Display text D	Text(32B),disp time(1B) Text(32B),disp time(1B) Text(32B),disp time(1B) Text(32B),disp time(1B)	017A -> 0199, 0153 019A -> 01B9, 0154 01BA -> 01D9, 0155 01DA -> 01F9, 0156
Jump to first (237)	Jump back to the first primitive	N/A	N/A
Synchronize (238)	Synchronization primitive	N/A	N/A
Sound A (239) Sound B (240)	Play sound A Play sound B	N/A N/A	N/A N/A



Rand. Amp. on (241)	Start random amplitude	Rand % (1B, ch1-4)	016F -> 0172
Rand. Amp. off (242)	Stop random amplitude	N/A	N/A
TTL trigger (243)	TTL Trigger	N/A	N/A
Push button sync (244)	Synchronized push button	N/A	N/A
Push button (245)	Push button (not synchronized)	N/A	N/A
Unused (246-247)	Not currently used	N/A	N/A
User Interaction A (248)	User Interaction A	User interaction A specifications	00C2 -> 00C7
User Interaction B (249)	User Interaction B	User interaction B specifications	00C8 -> 00CD
User Interaction C (250)	User Interaction C	User interaction C specifications	00CE -> 00D3
User Interaction D (251)	User Interaction D	User interaction D specifications	00D4 -> 00D9
User Interaction E (252)	User Interaction E	User interaction E specifications	00DA -> 00DF
User Interaction F (253)	User Interaction F	User interaction F specifications	00E0 -> 00E5
User Interaction G (254)	User Interaction G	User interaction G specifications	00E6 -> 00EB
End stimulation (255)	Ends stimulation for channel	N/A	N/A

**Table A4-2: Primitive encoding details.**

## Appendix 5: Experimental Results Timing Data

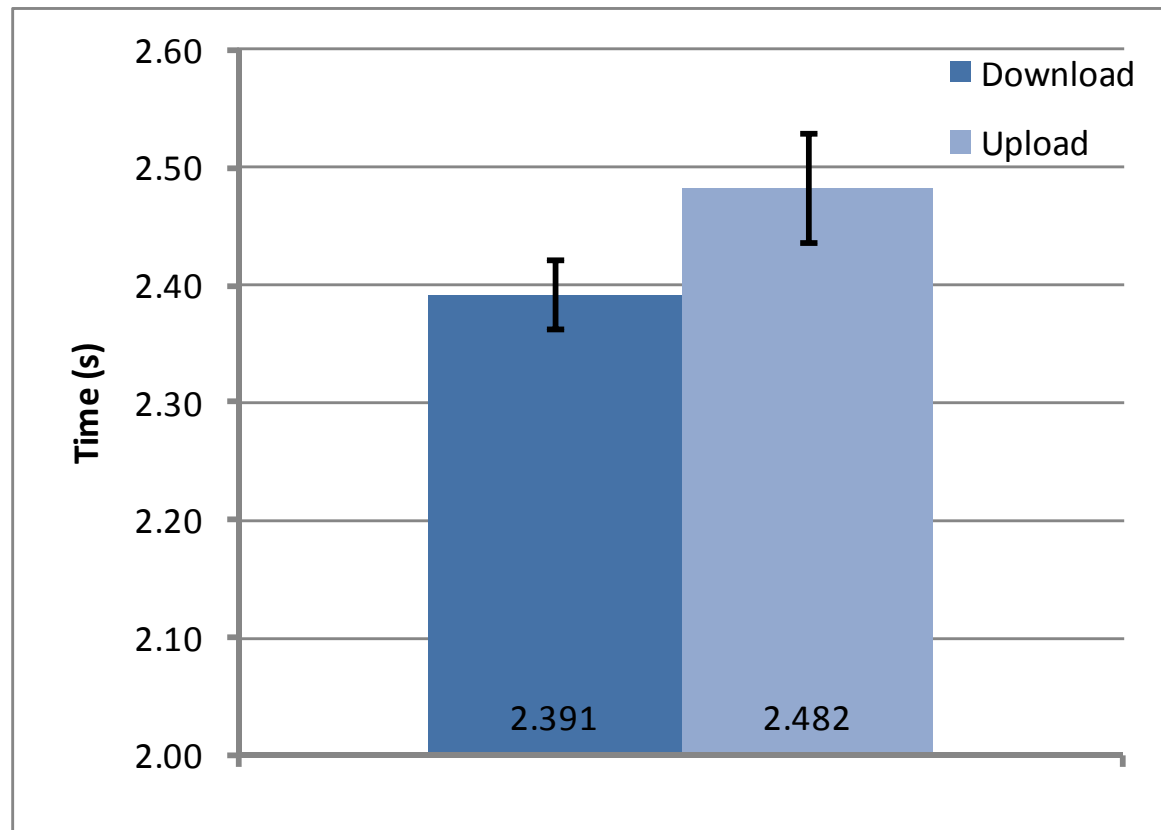
All timing data is based on a 12 MHz clock. The margin of errors are based on the standard deviation calculated from the sample data, sample size and a confidence coefficient of 0.95 (95%).

Legacy Protocol Download Time		
Sample	Clock cycles	Time (s)
1	28,921,750	2.410146
2	29,293,475	2.441123
3	28,958,116	2.413176
4	29,822,993	2.485249
5	29,990,373	2.499198
6	26,509,702	2.209142
7	28,621,380	2.385115
8	29,246,463	2.437205
9	28,827,576	2.402298
10	28,670,768	2.389231
11	28,429,831	2.369153
12	30,193,920	2.516160
13	28,945,881	2.412157
14	30,350,724	2.529227
15	29,809,561	2.484130
16	28,046,315	2.337193
17	28,705,813	2.392151
18	27,758,380	2.313198
19	28,814,194	2.401183
20	28,898,274	2.408190
21	29,870,746	2.489229
22	29,858,311	2.488193
23	27,469,958	2.289163
24	29,091,325	2.424277
25	27,301,918	2.275160
26	28,874,119	2.406177
27	27,458,047	2.288171
28	27,902,144	2.325179
29	26,906,596	2.242216
30	27,373,018	2.281085

Legacy Protocol Upload Time		
Sample	Clock cycles	Time (s)
1	32,174,431	2.681203
2	29,222,300	2.435192
3	29,953,690	2.496141
4	31,262,598	2.605217
5	27,927,171	2.327264
6	28,693,621	2.391135
7	28,826,057	2.402171
8	30,075,025	2.506252
9	29,498,663	2.458222
10	29,186,656	2.432221
11	29,234,330	2.436194
12	27,446,746	2.287229
13	27,661,588	2.305132
14	30,314,314	2.526193
15	27,637,693	2.303141
16	29,749,511	2.479126
17	30,614,875	2.551240
18	27,530,826	2.294236
19	29,174,129	2.431177
20	30,974,477	2.581206
21	30,434,654	2.536221
22	32,187,396	2.682283
23	30,230,238	2.519187
24	34,370,073	2.864173
25	29,954,856	2.496238
26	31,179,140	2.598262
27	28,574,838	2.381237
28	31,839,063	2.653255
29	28,958,261	2.413188
30	28,669,630	2.389136

Download Stats	Value
Sample size	30
Sample mean (s)	2.391
Minimum (s)	2.209
Maximum (s)	2.529
Std. Dev (s)	0.084362
Margin of error (s)	0.030188085

Upload Stats	Value
Sample size	30
Sample mean (s)	2.482
Minimum (s)	2.287
Maximum (s)	2.864
Std. Dev (s)	0.133165
Margin of error (s)	0.04765152

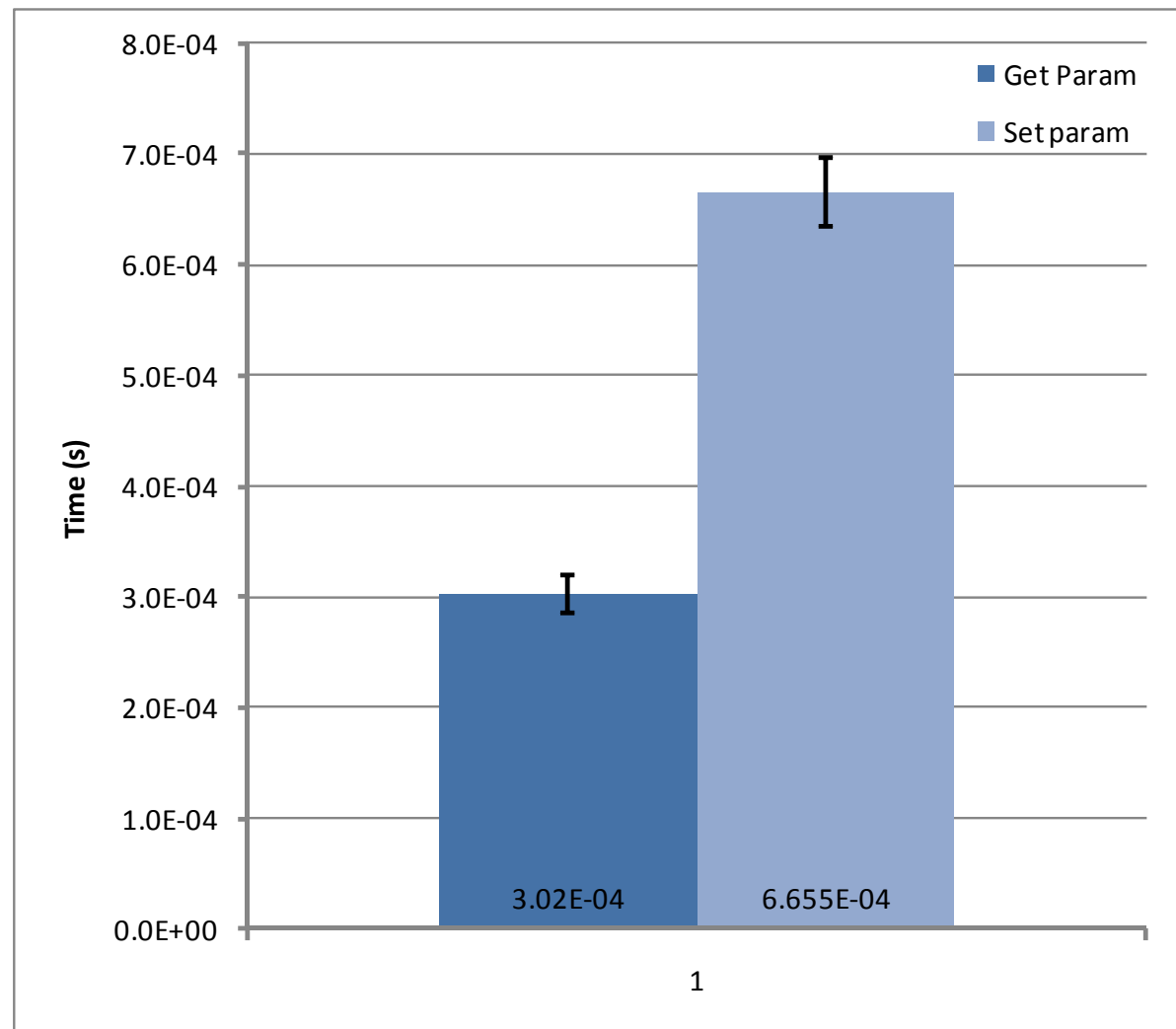


Get Parameter Time		
Sample	Clock cycles	Time (s)
1	4,478	3.731667E-04
2	3,090	2.575000E-04
3	4,682	3.901667E-04
4	3,256	2.713333E-04
5	3,307	2.755833E-04
6	3,268	2.723333E-04
7	3,392	2.826667E-04
8	3,272	2.726667E-04
9	4,624	3.853333E-04
10	4,801	4.000833E-04
11	3,393	2.827500E-04
12	3,849	3.207500E-04
13	3,105	2.587500E-04
14	3,091	2.575833E-04
15	3,114	2.595000E-04
16	3,862	3.218333E-04
17	3,057	2.547500E-04
18	3,088	2.573333E-04
19	3,700	3.083333E-04
20	3,264	2.720000E-04
21	3,071	2.559167E-04
22	4,756	3.963333E-04
23	3,427	2.855833E-04
24	3,483	2.902500E-04
25	3,410	2.841667E-04
26	4,831	4.025833E-04
27	3,517	2.930833E-04
28	3,314	2.761667E-04
29	3,407	2.839167E-04
30	3,866	3.221667E-04

Set Parameter Time		
Sample	Clock cycles	Time (s)
1	7,935	6.612500E-04
2	6,808	5.673333E-04
3	6,515	5.429167E-04
4	7,947	6.622500E-04
5	7,114	5.928333E-04
6	9,456	7.880000E-04
7	8,009	6.674167E-04
8	7,998	6.665000E-04
9	9,980	8.316667E-04
10	7,404	6.170000E-04
11	6,377	5.314167E-04
12	7,770	6.475000E-04
13	7,929	6.607500E-04
14	8,038	6.698333E-04
15	7,903	6.585833E-04
16	8,005	6.670833E-04
17	6,334	5.278333E-04
18	6,359	5.299167E-04
19	7,981	6.650833E-04
20	11,037	9.197500E-04
21	8,173	6.810833E-04
22	7,851	6.542500E-04
23	9,333	7.777500E-04
24	7,843	6.535833E-04
25	8,560	7.133333E-04
26	8,564	7.136667E-04
27	7,875	6.562500E-04
28	8,173	6.810833E-04
29	8,158	6.798333E-04
30	8,158	6.798333E-04

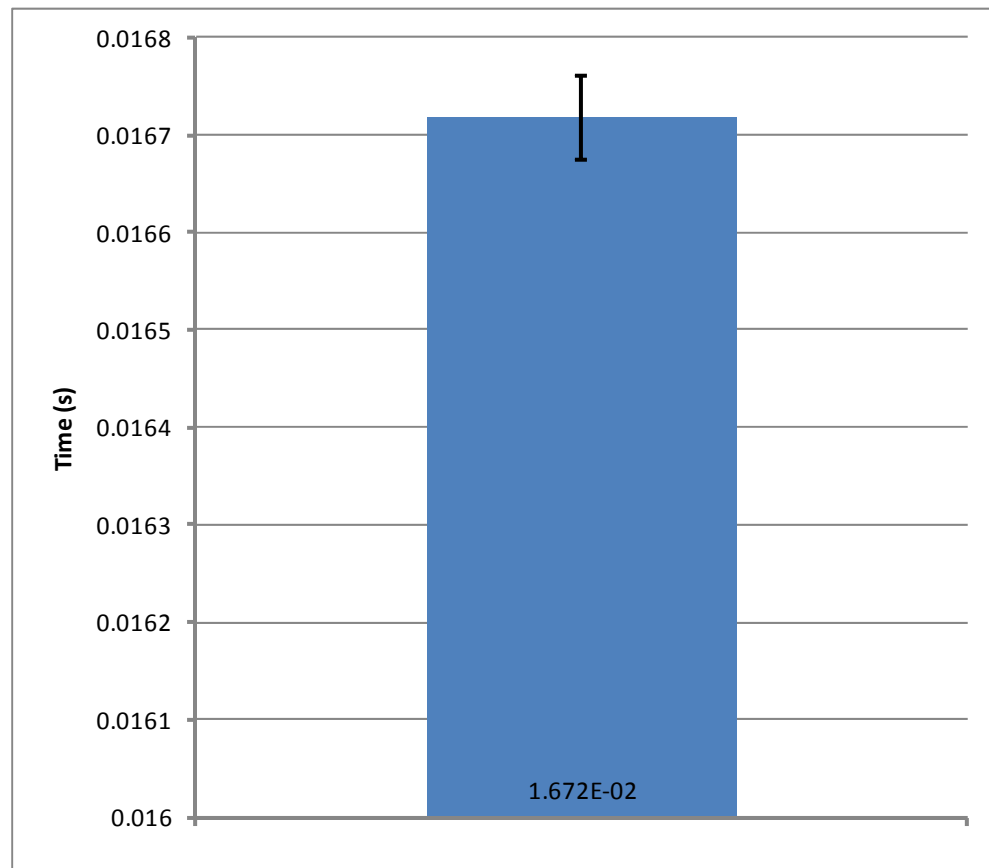
Get Param Stats	Value
Sample size	30
Sample mean (s)	3.02E-04
Minimum (s)	2.55E-04
Maximum (s)	4.03E-04
Std. Dev (s)	4.928E-05
Margin of error (s)	1.763E-05

Set Param Stats	Value
Sample size	30
Sample mean (s)	6.655E-04
Minimum (s)	5.278E-04
Maximum (s)	9.198E-04
Std. Dev (s)	8.586E-05
Margin of error (s)	3.072E-05



Change Stimulation Protocol Time		
Sample	Clock cycles	Time (s)
1	202,292	1.685767E-02
2	203,910	1.699250E-02
3	200,728	1.672733E-02
4	200,661	1.672175E-02
5	198,192	1.651600E-02
6	202,159	1.684658E-02
7	199,829	1.665242E-02
8	199,437	1.661975E-02
9	200,747	1.672892E-02
10	200,775	1.673125E-02
11	201,062	1.675517E-02
12	197,335	1.644458E-02
13	202,465	1.687208E-02
14	200,830	1.673583E-02
15	201,574	1.679783E-02
16	202,023	1.683525E-02
17	199,547	1.662892E-02
18	200,344	1.669533E-02
19	200,716	1.672633E-02
20	199,317	1.660975E-02
21	202,217	1.685142E-02
22	200,760	1.673000E-02
23	201,122	1.676017E-02
24	197,701	1.647508E-02
25	200,667	1.672225E-02
26	199,315	1.660958E-02
27	200,876	1.673967E-02
28	200,744	1.672867E-02
29	198,983	1.658192E-02
30	201,833	1.681942E-02

Sample size	30
Sample mean (s)	1.672E-02
Minimum (s)	1.644E-02
Maximum (s)	1.699E-02
Std. Dev (s)	1.213E-04
Margin of error (s)	4.342E-05

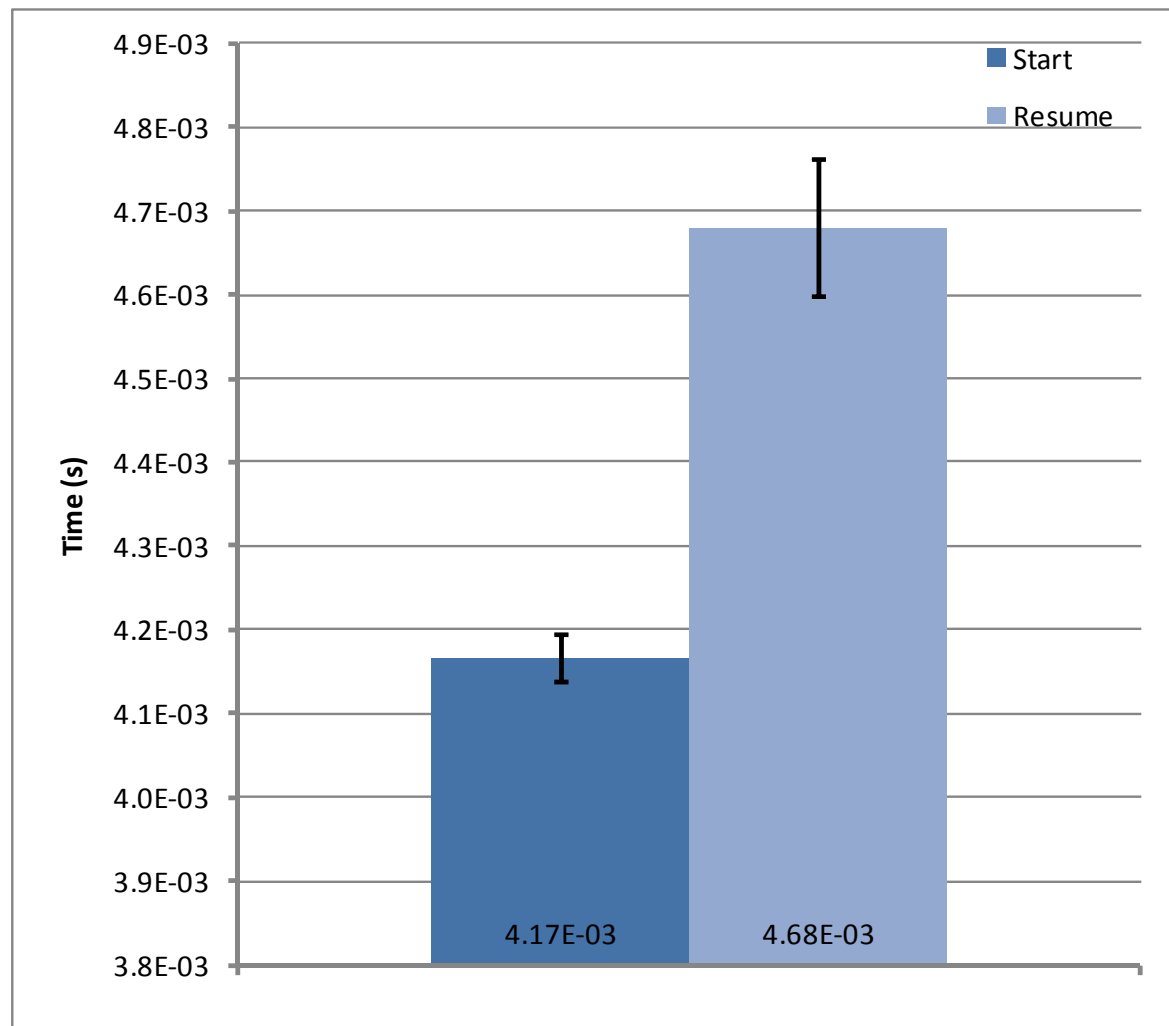


Start Stimulation Time		
Sample	Clock cycles	Time (s)
1	48,341	4.028417E-03
2	49,418	4.118167E-03
3	50,213	4.184417E-03
4	49,055	4.087917E-03
5	49,006	4.083833E-03
6	49,582	4.131833E-03
7	49,934	4.161167E-03
8	50,986	4.248833E-03
9	49,692	4.141000E-03
10	51,384	4.282000E-03
11	48,194	4.016167E-03
12	50,955	4.246250E-03
13	51,095	4.257917E-03
14	49,654	4.137833E-03
15	49,887	4.157250E-03
16	49,876	4.156333E-03
17	51,242	4.270167E-03
18	49,312	4.109333E-03
19	50,319	4.193250E-03
20	50,041	4.170083E-03
21	48,468	4.039000E-03
22	48,974	4.081167E-03
23	51,245	4.270417E-03
24	50,775	4.231250E-03
25	50,110	4.175833E-03
26	50,899	4.241583E-03
27	50,109	4.175750E-03
28	51,350	4.279167E-03
29	49,013	4.084417E-03
30	50,837	4.236417E-03

Resume Stimulation Time		
Sample	Clock cycles	Time (s)
1	62,976	5.248000E-03
2	58,948	4.912333E-03
3	59,859	4.988250E-03
4	58,287	4.857250E-03
5	59,412	4.951000E-03
6	56,315	4.692917E-03
7	55,447	4.620583E-03
8	63,797	5.316417E-03
9	57,361	4.780083E-03
10	58,724	4.893667E-03
11	58,069	4.839083E-03
12	54,184	4.515333E-03
13	55,341	4.611750E-03
14	55,265	4.605417E-03
15	54,121	4.510083E-03
16	55,008	4.584000E-03
17	54,630	4.552500E-03
18	54,985	4.582083E-03
19	54,357	4.529750E-03
20	54,337	4.528083E-03
21	53,931	4.494250E-03
22	54,431	4.535917E-03
23	54,456	4.538000E-03
24	54,764	4.563667E-03
25	54,885	4.573750E-03
26	54,043	4.503583E-03
27	56,100	4.675000E-03
28	52,598	4.383167E-03
29	52,658	4.388167E-03
30	54,913	4.576083E-03

Start Stats	Value
Sample size	30
Sample mean (s)	4.17E-03
Minimum (s)	4.02E-03
Maximum (s)	4.28E-03
Std. Dev (s)	7.845E-05
Margin of error (s)	2.807E-05

Resume Stats	Value
Sample size	30
Sample mean (s)	4.68E-03
Minimum (s)	4.38E-03
Maximum (s)	5.32E-03
Std. Dev (s)	2.296E-04
Margin of error (s)	8.215E-05



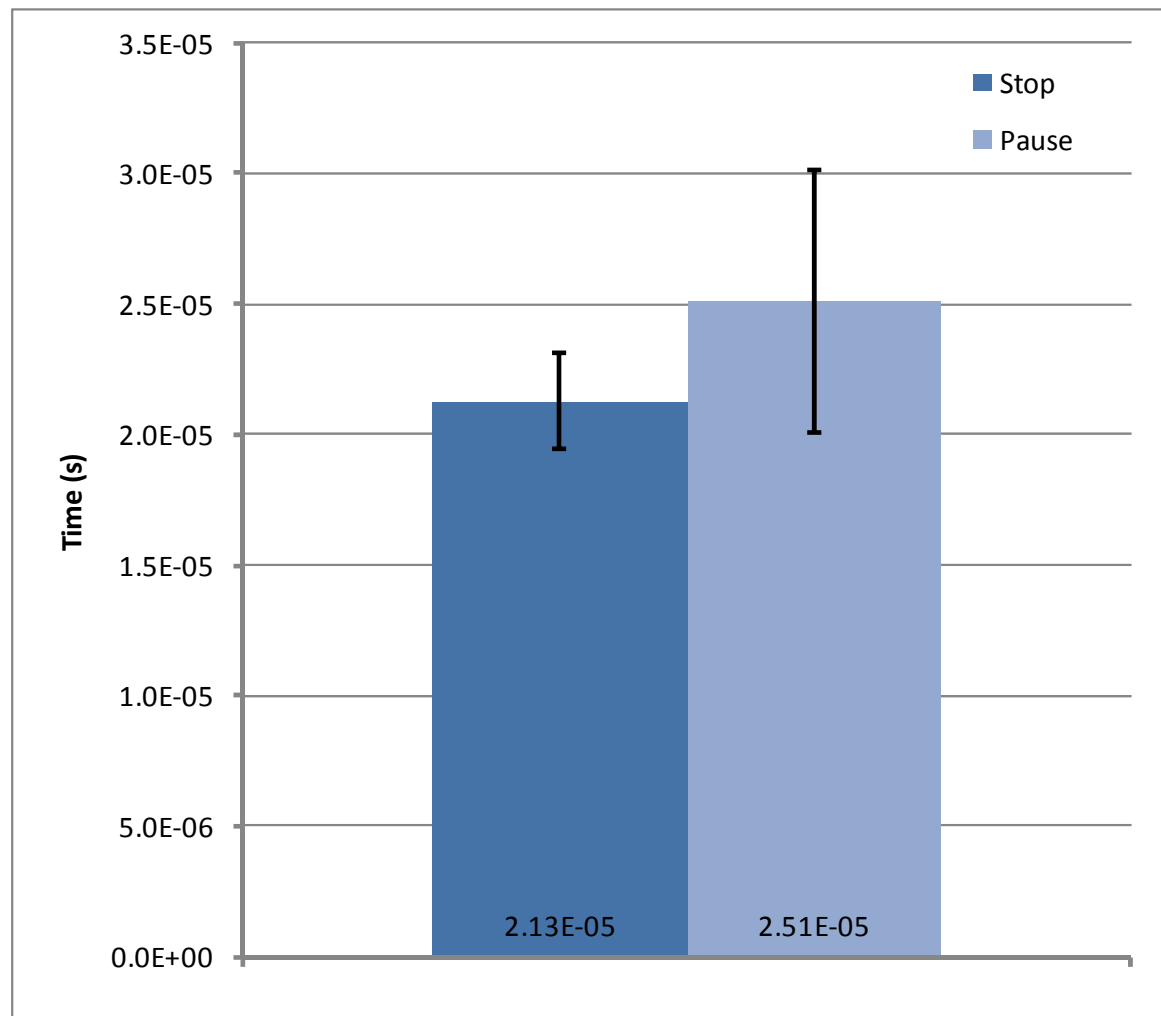


Stop Stimulation Time		
Sample	Clock cycles	Time (s)
1	225	1.875000E-05
2	225	1.875000E-05
3	225	1.875000E-05
4	377	3.141667E-05
5	376	3.133333E-05
6	225	1.875000E-05
7	225	1.875000E-05
8	225	1.875000E-05
9	225	1.875000E-05
10	225	1.875000E-05
11	225	1.875000E-05
12	377	3.141667E-05
13	225	1.875000E-05
14	225	1.875000E-05
15	225	1.875000E-05
16	225	1.875000E-05
17	377	3.141667E-05
18	225	1.875000E-05
19	225	1.875000E-05
20	377	3.141667E-05
21	225	1.875000E-05
22	225	1.875000E-05
23	377	3.141667E-05
24	225	1.875000E-05
25	225	1.875000E-05
26	225	1.875000E-05
27	225	1.875000E-05
28	225	1.875000E-05
29	225	1.875000E-05
30	225	1.875000E-05

Pause Stimulation Time		
Sample	Clock cycles	Time (s)
1	373	3.108333E-05
2	223	1.858333E-05
3	828	6.900000E-05
4	223	1.858333E-05
5	373	3.108333E-05
6	223	1.858333E-05
7	373	3.108333E-05
8	223	1.858333E-05
9	223	1.858333E-05
10	373	3.108333E-05
11	373	3.108333E-05
12	223	1.858333E-05
13	223	1.858333E-05
14	373	3.108333E-05
15	223	1.858333E-05
16	223	1.858333E-05
17	223	1.858333E-05
18	223	1.858333E-05
19	223	1.858333E-05
20	223	1.858333E-05
21	223	1.858333E-05
22	921	7.675000E-05
23	223	1.858333E-05
24	223	1.858333E-05
25	373	3.108333E-05
26	223	1.858333E-05
27	223	1.858333E-05
28	223	1.858333E-05
29	223	1.858333E-05
30	223	1.858333E-05

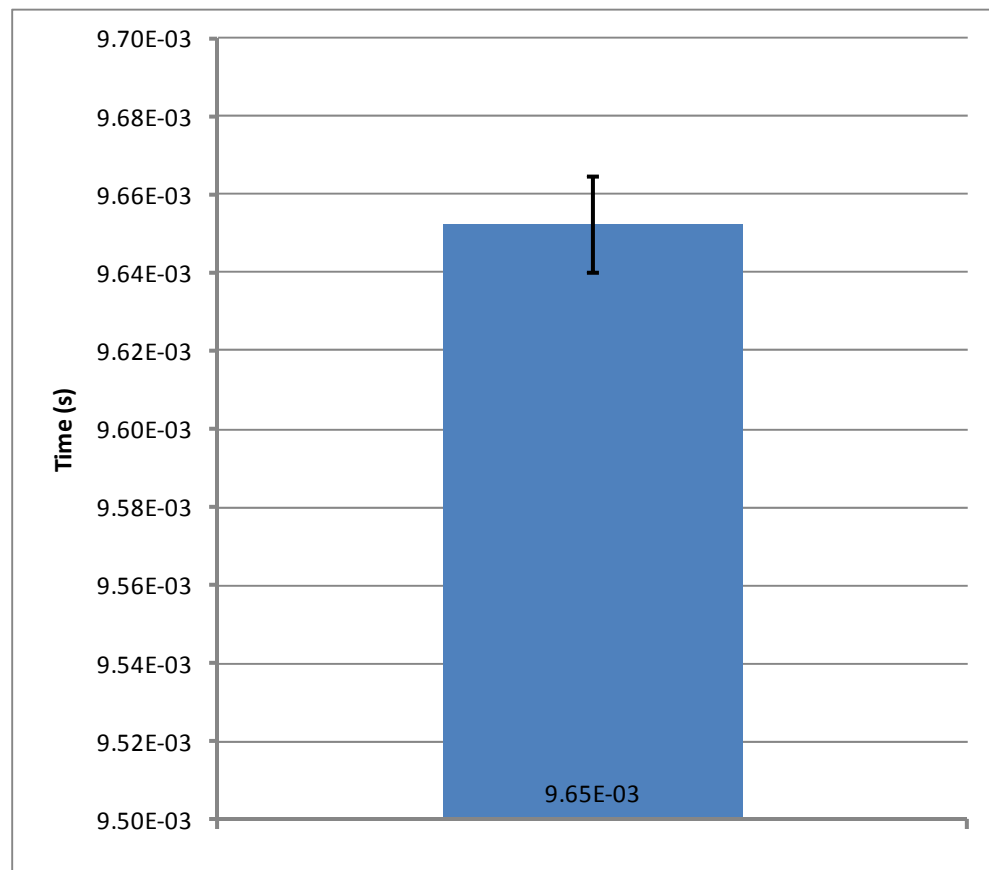
Stop Stats	Value
Sample size	30
Sample mean (s)	2.13E-05
Minimum (s)	1.88E-05
Maximum (s)	3.14E-05
Std. Dev (s)	5.148E-06
Margin of error (s)	1.842E-06

Pause Stats	Value
Sample size	30
Sample mean (s)	2.51E-05
Minimum (s)	1.86E-05
Maximum (s)	7.68E-05
Std. Dev (s)	1.407E-05
Margin of error (s)	5.033E-06



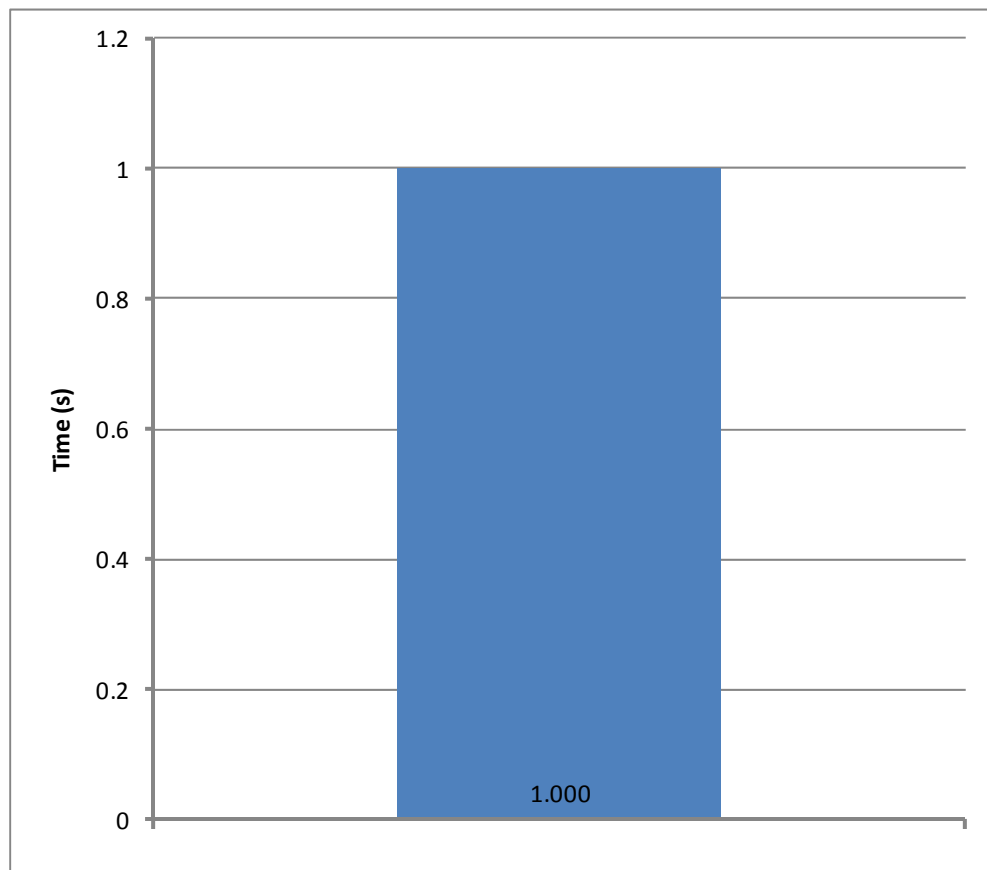
Prot. Processing Time (Worst Case)		
Sample	Clock cycles	Time (s)
1	115,970	9.664167E-03
2	115,729	9.644083E-03
3	116,002	9.666833E-03
4	115,872	9.656000E-03
5	116,153	9.679417E-03
6	115,670	9.639167E-03
7	115,990	9.665833E-03
8	115,981	9.665083E-03
9	115,982	9.665167E-03
10	116,176	9.681333E-03
11	114,907	9.575583E-03
12	116,174	9.681167E-03
13	115,247	9.603917E-03
14	115,713	9.642750E-03
15	116,183	9.681917E-03
16	115,676	9.639667E-03
17	116,015	9.667917E-03
18	116,177	9.681417E-03
19	115,728	9.644000E-03
20	114,220	9.518333E-03
21	115,996	9.666333E-03
22	116,174	9.681167E-03
23	115,744	9.645333E-03
24	116,021	9.668417E-03
25	116,195	9.682917E-03
26	115,669	9.639083E-03
27	115,713	9.642750E-03
28	115,998	9.666500E-03
29	116,030	9.669167E-03
30	115,726	9.643833E-03

Process Stats	Value
Sample size	30
Sample mean (s)	9.65E-03
Minimum (s)	9.52E-03
Maximum (s)	9.68E-03
Std. Dev (s)	3.479E-05
Margin of error (s)	1.245E-05



Stim. Trigger Timer (10x0.1s)		
Sample	Clock cycles	Time (s)
1	12,000,010	1.000001
2	12,000,010	1.000001
3	12,000,010	1.000001
4	12,000,010	1.000001
5	12,000,010	1.000001
6	12,000,010	1.000001
7	12,000,010	1.000001
8	12,000,010	1.000001
9	12,000,010	1.000001
10	12,000,010	1.000001
11	12,000,010	1.000001
12	12,000,010	1.000001
13	12,000,010	1.000001
14	12,000,010	1.000001
15	12,000,010	1.000001
16	12,000,010	1.000001
17	12,000,010	1.000001
18	12,000,010	1.000001
19	12,000,010	1.000001
20	12,000,010	1.000001
21	12,000,010	1.000001
22	12,000,010	1.000001
23	12,000,010	1.000001
24	12,000,010	1.000001
25	12,000,010	1.000001
26	12,000,010	1.000001
27	12,000,010	1.000001
28	12,000,010	1.000001
29	12,000,010	1.000001
30	12,000,010	1.000001

Trigger timer Stats	Value
Sample size	30
Sample mean (s)	1.000
Minimum (s)	1.000
Maximum (s)	1.000
Std. Dev (s)	6.775E-16
Margin of error (s)	2.424E-16



TTL Response Time		
Sample	Clock cycles	Time (s)
1	642	5.350000E-05
2	490	4.083333E-05
3	490	4.083333E-05
4	490	4.083333E-05
5	490	4.083333E-05
6	643	5.358333E-05
7	663	5.525000E-05
8	490	4.083333E-05
9	521	4.341667E-05
10	642	5.350000E-05
11	496	4.133333E-05
12	490	4.083333E-05
13	490	4.083333E-05
14	490	4.083333E-05
15	490	4.083333E-05
16	490	4.083333E-05
17	490	4.083333E-05
18	490	4.083333E-05
19	496	4.133333E-05
20	490	4.083333E-05
21	643	5.358333E-05
22	641	5.341667E-05
23	490	4.083333E-05
24	490	4.083333E-05
25	653	5.441667E-05
26	490	4.083333E-05
27	549	4.575000E-05
28	642	5.350000E-05
29	490	4.083333E-05
30	490	4.083333E-05

TTL Response Stats	Value
Sample size	30
Sample mean (s)	4.46E-05
Minimum (s)	4.08E-05
Maximum (s)	5.53E-05
Std. Dev (s)	5.771E-06
Margin of error (s)	2.065E-06

