NTNU
Innovation and Creativity

Communication Protocol for Advanced Prosthesis Components

David Karnå

Master of Science in Engineering Cybernetics
Submission date: June 2007
Supervisor: Geir Mathisen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

Det foreligger planer for en standard strømforsynings- og kommunikasjonsprotokoll for protesekomponenter, noe som vil forenkle sammenkopling av hender, albuer m.m. fra ulike leverandører samt tillate mer avanserte, koordinerte styreprinsipper. Standarden går under det foreløpige navnet Standardised Communication Interface for Prosthetics - SCIP.
Det foreligger en foreløpig funksjonsspesifikasjon for SCIP. Denne oppgaven går ut på å kartlegge ulike eksisterende lavnivå protokoller som vil kunne oppfylle funksjonsspesifikasjonen, samt utvikle,
realisere og evaluere utvalgte elementer av protokollen for en av de aktuelle lavnivåprotokollene.
1. Gi en oversikt over lavnivåprotokoller som du mener er relevante for den aktuelle anvendelsen. Legg spesielt vekt på forhold som er av direkte eller indirekte betydning for den foreliggende funksjonsspesifikasjonen.
2. Velg ut en av protokollene, og spesifiser en høynivåprotokoll basert på denne som tilfredsstiller de funksjonelle kravene som er stilt til SCIP.
3. Implementer deler av eller hele protokollen, og test, så langt tiden tillater det, hvorvidt den spesifiserte oppførselen er oppnådd.
4. Foreslå endringer og/eller tilføyelser til den foreliggende funksjonsspesifikasjonen i den grad du finner dette nødvendig eller naturlig.

Assignment given: 08. January 2007
Supervisor: Geir Mathisen, ITK

Preface

This master thesis is as I mention in the introduction just a part of an interesting project that aims towards defining an international standard for communication in powered upper-limb prostheses. I hope that this report can contribute a little to that work, and that other students wish to continue where I let go.

It has been very instructive to work with this kind of formal design, and I honestly believe that I have learnt much during this spring.

I would like to thank my advisor Øyvind Stavdahl for his guidance. His positive attitude, and sincere enthusiasm has been very inspiring. I would also like to thank my classmates, Morten Engen and Anders Fougner who have been great to discuss my work with. Finally I would like to thank NTNU for the privilege it has been to study here.

Trondheim 19/6-07

David Karnå

Abstract

It would be of great value for the prosthesis industry to achieve an open standard for communication in upper limb prostheses. Cooperation between NTNU and the University of New Brunswick has resulted in a functional requirements specification for such a standard, SCIP(Standardised Communication Interface in Prostheses). The special challenges for communication in a prosthesis system are possible noisy environments, high demands for light weight, safety for the user and the fact that devices might be switched during operation.

It was the purpose of this master thesis to make a design based upon those requirements. This was done by first choosing an existing bus standard, that would provide the lower levels of communication. CAN was chosen for this purpose. The next step of the design process was to transform the functional requirements into more specific technical requirements. This resulted in the definition of four types of nodes on the bus. These are bus controller, input controller, device controller and service controller. Their interactions called for the specification of several different message types, to support data exchange between the nodes.

The result was a design that specifies node types, message types, variables like adresses, control strategies etc., state transition diagrams for the different node types and some message sequences. It also specifies the use of the CAN data-frame for all message types.

List of Figures

Contents

1 Introduction

In the area of powered upper limb prostheses improvements are continously made, as new technology allows devices to be made lighter and easier to control, battery lifetime is prolonged and our understanding of the interaction between man and machine is increasing. A cooperation project going on between NTNU and The University of New Brunnswick among others that aims to contribute to these improvements is SCIP. SCIP or Standardised Communication Interface in Prostheses is meant to become an open standard for data communication in upper limb prostheses. The advancements achieved with a bus solution like this are increased modularity, lighter and more robust prostheses as a result of reduced wiring and possibility for centralized control.

This master thesis is a part of the SCIP project, and is based upon a functional requirements specification written by Øyvind Stavdahl, Peter J. Kyberd and Geir Mathisen. The task is to design a solution for SCIP built upon an available low level bus protocol. This should then be implemented as far as time suffices.

First a decision must be made as to which bus protocol to use, and this is the topic of chap.3. After that the functional requirements specification is rewritten in terms of more concrete technical requirements in chap.4. Chapter 5 describes the behaviour of the nodes in the network, and specifies the messages needed to fulfill the technical requirements. The layout of these messages and specification of the use of variable values also belong in that chapter. Next, chap.6 provides a critical review of the design choices, and chap.7 states the authors view of the final design. Finally chap.8 gives some suggestions to what could and should be done next. Some relevant theory is presented in chap.2

2 Background

This chapter will introduce a few important concepts and standards which are much used in the area of networking. Following that is a short introduction to the anatomy of the human upper limb. The last section provides some information about control strategies in prostheses and the input signals used for this control.

2.1 Networking Concepts and Standards

2.1.1 ISO/OSI

This is a standardized model which divides a network into seven sub-layers. The sub-layers are listed here in top-down order:

- 7. Application Layer
- 6. Presentation Layer
- 5. Session Layer
- 4. Transport Layer
- 3. Network Layer
- 2. Data Link Layer
- 1. Physical Layer

The following explanation is a short version of the one found in (Tanenbaum 2003, chap.1.4) The application layer defines the interface to the end-user, i.e. what data to send and what to do with data received. The presentation layer deals with converting the structure of the data. This is only necessary if the application layer uses data structures that are non standard. The session layer allows two machines to establish sessions. The session might set restrictions with regard to who can send data when, and might also provide checkpointing. The transport layer provides end to end communication by breaking down user data to smaller packets handled in turn by the network layer. The network layer is responsible for routing, and thus decides the quality of service provided(for example delay). The data link layer handles even smaller packets of data called data frames. These may contain error handling information in addition to data and adresses. Finally the physical layer transmits raw bits over some communication media.

2.1.2 CSMA/CD

Carrier sense multiple access/Collision detection is a standard for buses with several nodes that can initiate communication. CSMA means quite simply that each node scans the bus for a short period of inactivity before starting a transmission. CD requires that the nodes can read the bit value on the bus while sending. If discovering an inconsistency, the node simply stops the transfer. The node then takes the action specified by the bus standard. The solution used by ethernet is to wait for a random time, then retransmit the data. For more information on this topic (see Onshus 2006)

2.1.3 Network Topologies

There are a number of different basic network topologies and hybrids that mix those. The most common basic topologies are bus, star, ring and tree according to (Olsen 1998, chap.4). The only ones mentioned in this report are bus and star. They are illustrated in fig.1. The only difference between an ordinary star topology and the tiered star is that the ordinary one does not have a root node.

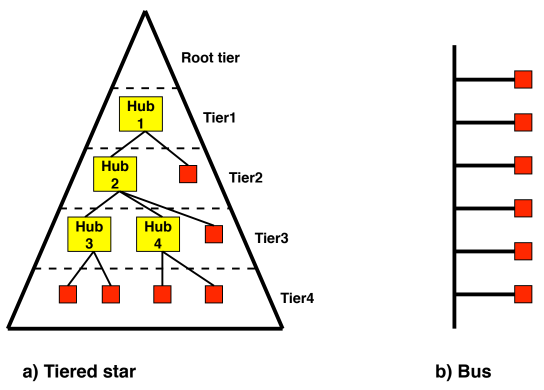

Figure 1: Network topologies

The main advantage of a star topology over a bus is that a broken wire does not affect the entire network. The main advantages of a bus topology are reduced wiring, and the lack of need for hubs.

2.2 Basic Anatomy of The Upper Limb

This chapter gives only a very brief introduction to the anatomy of the human upper limb. For a more detailed description (see Øyvind Stavdahl 2002, Chap. 2)

2.2.1 The Standard Anatomical Position

In fig.2 the standard anatomical position is illustrated. Also some terms that refer to different directions relative to the human body are specified. Proximal means closer to the trunk, and distal is the opposite. Dorsal means the back of the body and ventral means the front. Lateral and medial refer to further from or closer to the plane that divides the body in two symmetrical parts.

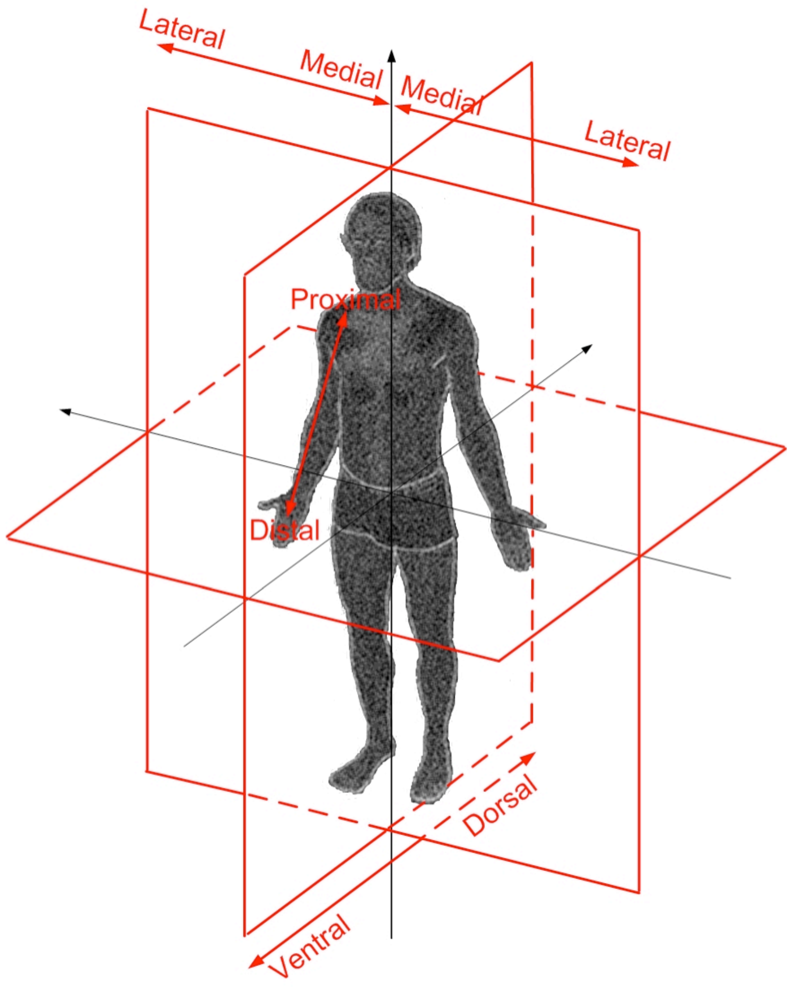

Figure 2: The Standard Anatomical Position and relative directions

2.2.2 The Shoulder Joint

A human shoulder has three degrees of freedom(DoF). These are flexion/extension, pronation/supination and abduction/adduction. Flexion refers to ventral displacement of the arm, and extension refers to dorsal displacement. Pronation and supination refer to rotation of the arm about the long axis of the arm. Pronation rotates the palm medially, while supination rotates the palm laterally. Finally abduction means lateral displacement of the arm, and adduction means medial displacement. All explanations above refer to movements starting in the standard anatomical position of fig.2.

2.2.3 The Elbow Joint

A human elbow exhibits only one DoF, flexion/extension. Flexion refers to bending the elbow and extension naturally means extending the elbow.

2.2.4 The Wrist Joint

A human wrist exhibits the same three degrees of freedom as the shoulder. This is illustrated in fig.3. Note that adduction is termed ulnar deviation and abduction is termed radial deviation in the figure. These terms are used throughout this report for wrist movements, and the corresponding DoF is called radioulnar deviation.

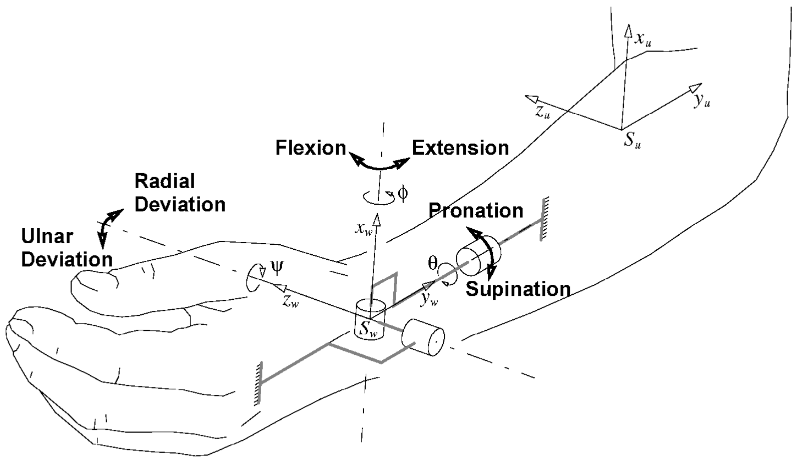

Figure 3: The degrees of freedom of the human wrist joint(Courtesy to Øyvind Stavdahl)

2.2.5 The Hand and Fingers

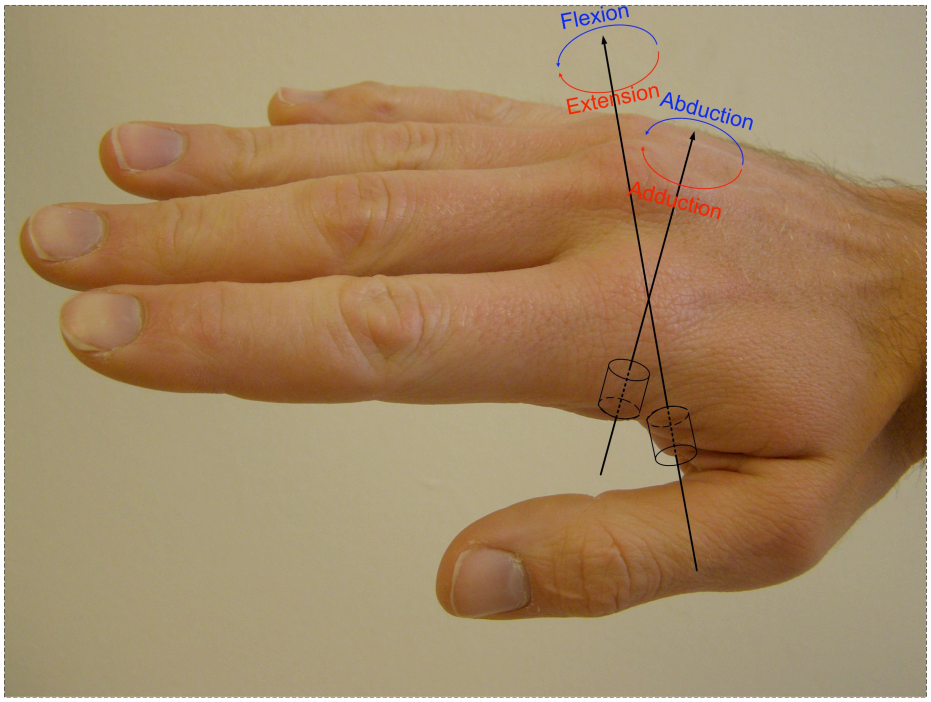

Figure 4: The degrees of freedom of the metacarpophalangeal joint of the index finger.

The fingers of a human hand has four degrees of freedom each. The first three are flexion/extension of the metacarpophalangeal(MCP) joint(see fig.4), the proximal interphalangeal joint(PIP) and the distal interphalangeal joint(DIP). The last one is abduction/adduction of the MCP. The thumb is slightly more complex with five DoFs. They are flexion/extension of the carpometacarpal(CMC) joint, the MCP joint and the interphalangeal(IP) joint. In addition the thumb supports abduction/adduction of the CMC joint and the MCP joint. This adds up to a total of 21 degrees of freedom for the whole hand. The joint placement within the hand is shown in fig.5 More detail on this hand model is found in the article(Rijpkema & Girard 1991).

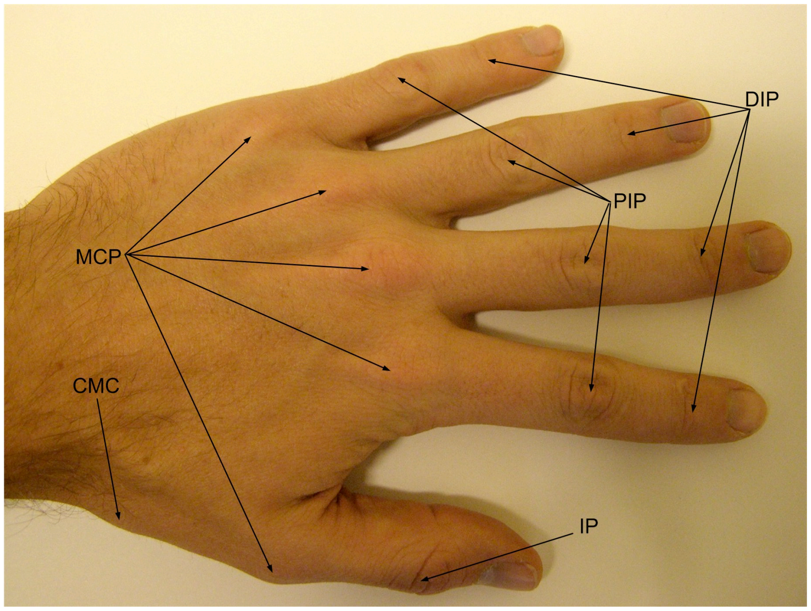

Figure 5: The joints of the human hand.

2.3 Control strategies in prostheses

This chapter will describe the most common control strategies which are used in todays prostheses, and also the sensors used to register input from the user. For more information on control strategies (see Muzumdar 2004, chap.3)

2.3.1 Sensor types

The most important sensor types in use in todays prostheses are switches, myoelectric sensors, variable resistors(f.ex. force sensing resistors(FSR)) and linear potentiometers. For more details on these sensors (see Karnå 2006, chap.2). Switches give digital signals as output, while all the other types produce analog signals. The resolution of these signals depend on the A/D converter used, but 10 bit and 16 bit are common.

It is important to separate between raw(only amplified and rectified) myoelectric signals(MES) and processed(amplified, rectified and smoothed) myoelectric signals(PMES). In a control system where raw signals are used the sampling rate must be at least 1kHz. This is due to the fact that MES includes frequency components of up to 500 Hz.[1] If smoothed signals are used the control strategy dictates the sampling frequency, which should not be below 20Hz.

[1]It actually contains higher frequency components, but these are small enough that downfolding of them will not distort the signal notably.

2.3.2 Single-Site Systems

In a single site system only one input is used for controlling one degree of freedom(DoF). Any of the sensors described above could be used to generate this input, except for a switch. There are two commonly used strategies for single site control. These are illustrated for a hand device in fig.6 and fig.7. The first figure shows level coding. It is achieved by separating between three levels of the input. Typically the lowest level would give no motor output, while the two higher levels would rotate the device in opposite directons.

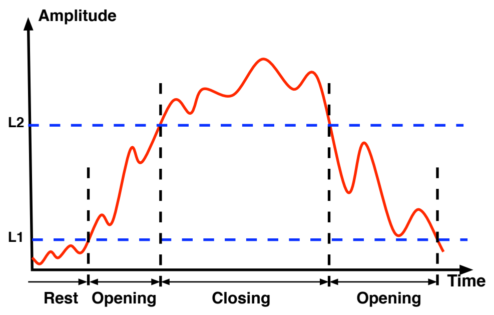

Figure 6: Single-site control by use of level coding.

The second method is called rate coding. It uses the average rate of change of the input over a small time interval(decision time) to decide the direction of rotation. The decision time is constant, and the count starts when the signal amplitude exceeds the theshold level(L1). A quick change in level(fig.7a) sets one direction, and a slow change(fig.7b) the other. This direction of rotation is then maintained until the signal again drops below the threshold level(L1).

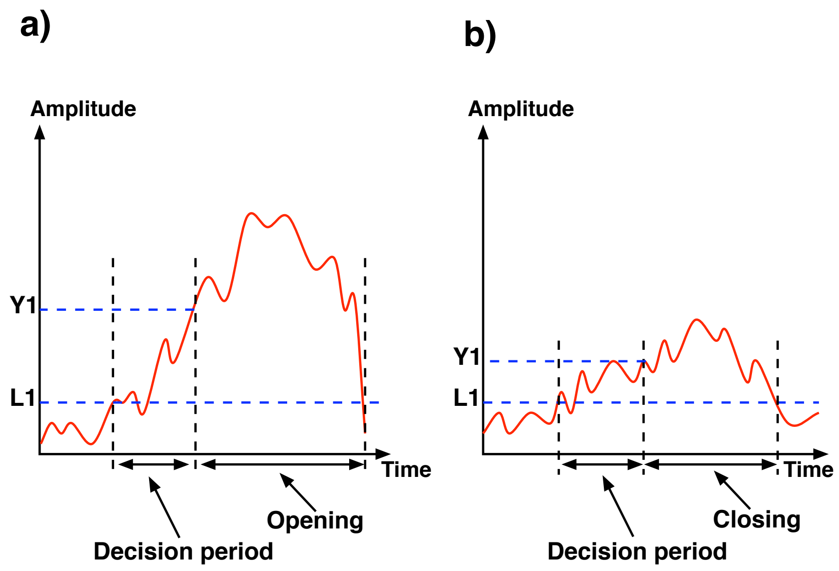

Figure 7: Single-site control by use of rate coding.

2.3.3 Two-Site Systems

This strategy uses two input for controlling one degree of freedom. For example in a wrist prosthesis one input could control flexion and the other extension. It is common to use myoelectric signals(MES) from remnant muscles as input, but one could also use two FSR:s or even two switch-signals.

2.3.4 Multi Function Control Strategies

The aim of these strategies is to control several functions with the same myoelectric system. One way that this could be done is to use raw MES and look at the pattern of the signals. This way more information could be extracted than in a traditional PMES based system, where only the level of the signal is considered. This was done in a swedish prosthesis called the Sven Hand. It uses two raw MES to control three degrees of freedom. To accomplish this six different patterns are recognized and used to control one direction of rotation of one DoF each.

Another kind of multi function control strategy has been implemented in the Southampton hand. It is based on a two-site myoelectric signal, to control up to five independent fingers. This control scheme uses touch and slip sensors on each finger to provide feedback to a microcontroller. The microcontroller then uses the two MES and the sensor feedback to decide on the best grip when holding an object.

3 Choice of Bus Solution

This chapter starts with a short description of each of the buses, which might provide the basis for SCIP. Following that is a listing of the SCIP requirements that should be fulfilled on this low level of the protocol, and whether or not the different buses provide solutions to these. The final section provides a short discussion on the final choice of bus standard for SCIP.

3.1 Bus Standards

The first two subsections of this section describe Wireless data transfer and Paralell buses respectively. The following nine subsections all describe different serial bus standards starting with the simplest, and then gradually moving up in complexity. Most serial standards support only bus topology(see chap.2.1.3). For the standards where this is not the case, the supported topology/topologies is/are mentioned. Most bus standards are described in more detail in (Catsoulis 2003)

3.1.1 Wireless Data Transfer

The two most used wireless standards for short range, low power data transfer are Bluetooth and ZigBee. Bluetooth is a high bandwidth standard which unfortunately consumes a lot of power. ZigBee on the other hand has low power consumption, but the bandwidth is only 250kbps. This is probably a little bit low for SCIP, especially for a wireless standard where retransmission due to corruption is more probable. ZigBee and Bluetooth are therefore not very well suited for use in SCIP.

There is another wireless standard under development which might provide a realistic alternative for SCIP. It is called WiBree and has a bandwidth of 1Mbps. Also it is supposed to have much lower power consumption than Bluetooth. This might be an interesting alternative for the future, but as the standard is not fully developed yet it will not be considered further in this report.

3.1.2 Parallel Buses

Parallel data buses naturally have very high bandwidth due to the ability of sending a whole data word, for example 8 bit simultaneously. The main drawback of a parallel bus is the greatly increased number of wires needed. This rules out a parallel bus as a solution for SCIP, as it is stated in the functional requirements specification that a maximum of two wires shall be used for data. Parallel buses are therefore not considered further in this report.

3.1.3 UART(Universal Asynchronous Receiver Transmitter)

UART is a very simple serial transfer protocol, with one channel for receiving and one for transmitting data. Being asynchronous it requires transfer rates, or baud rates as it is often called, to be hardcoded in the sending and receiving devices. Further UART is best suited for 1 to 1 communication, as a device in a UART network would have no way of detecting another device sending at the same time. It provides simple error detection through parity bit stuffing.

3.1.4 SPI(Serial Peripheral Interface)

This is another simple protocol, which uses (minimum) four wires for communication. It is a single master multiple slave protocol, and one of the wires(SS) is used for selecting the slave which the master wants to communicate with. In normal operation there is one SS for each slave, making the SPI unsuited for large systems. The other wires are data lines(Master in slave out(MISO) and Master out slave in(MOSI)), and a clock(SCL) for synchronization.

3.1.5 I2C(Inter Integrated Circuit)

I2C, or TWI(Two Wire Interface) as it is sometimes called, uses the two wires SDA(serial data) and SCL(serial clock) to connect devices. It uses adressed communication, and supports broadcasting of only a few predefined messages, for example reset. The frame size is 21 bit including 8 bit data, and each device could act as master or slave.

3.1.6 OneWire

OneWire has as the name suggests one very desirable feature, namely only one wire for both data transfer and power(It naturally needs one extra wire for ground). It uses very long adresses(8bytes), and the idea behind that is to have statical adresses without running the risk of two nodes sharing adress.

3.1.7 CAN(Controller Area Network)

CAN is a very robust bus standard which is, for that reason, much used in automotive applications. It is message based meaning that every node can read every message on the bus. Further it uses the CSMA/CD standard described in chap.2.1.2 for bus access. The robustness is due to the facts that CAN sends data over a differential wire pair(see chap.3.2.4), and that it implements automatic retransmit upon error discovery together with CRC coding(see 3.2.9). For more information on CAN see (Pazul 1999).

3.1.8 TTP(Time Triggered Protocol)

TTP differs from all other protocols mentioned here in that it divides the bandwidth of the bus into time slots. Every node has its unique time slot where only it is allowed to send. Hardware bus-guardians make sure that no nodes violate this scheme, and thus a babbling idiot can not prevent other nodes from using the bus[2]. TTP is well suited for time critical applications, since one can always predict how often a node gets access to the bus. TTP supports both star and

[2]An exception is the rare case where both the node and its bus guardian fail simultaneously

bus topologies. (Kopetz 1998) gives a good introduction to the special features of TTP.

3.1.9 USB(Universal Serial Bus)

This is a very widespread bus standard, used primarily in PC:s but also in other local networks. It has a tiered star topology(see fig.1). Like CAN USB transmits data over a differential wire pair. In a USB network there is one and only one host, which is responsible for initiating all communication. The maximum number of peripherals is 127. For USB communication is more complex than for the bus types mentioned above. One data transmission actually consists of three parts. Initially a token packet is transmitted, telling what kind the next packet is. Then a data packet is transmitted and finally a handshaking packet is returned.

3.1.10 FireWire

FireWire is a rather complex bus standard which supports two different modes of communication. These are asynchronous and isochronous. In isochronous mode messages are broadcast on a communication channel without any form of acknowledment. This is well suited for time critical data transfer like real time audio or video. Asynchronous transfer would be the mode of choice for SCIP, as this means adressed messages with acknowledgement. For further reading see(Wickelgren 1997)

3.1.11 Ethernet

Ethernet is not a simple standard, but a family of several standards differing in speed, network topology, transportation medium etc. Star topology is the most commonly used network topology, and all ethernet standards are fast with the slowest one running at 10Mb/s. Ethernet uses CSMA/CD for collision handling, and implements either full- or half duplex allowing nodes to transmit and receive data simultaneously.

3.2 Requirements for SCIP

There are many requirements for SCIP that will not be mentioned in this chapter. In fact the requirements mentioned in this chapter are just the ones that should preferrably be placed in one of the lowest two layers of the ISO/OSI model. The reason for this is that SCIP will only need to implement layer 1,2 and 7 of ISO/OSI. Furthermore the bus protocols considered here only implement parts of layers 1 and 2. Thus layer 7, and possibly parts of layer 1 and 2, are left for the SCIP protocol to define.

3.2.1 Transfer Speed

By far the most time-critical information exchange which SCIP will have to deal with are raw myo electric signals(MES). It is common practice to set the highest measured frequency for MES to 500 Hz, and according to the Nyquist sampling theorem it will then be necessary to sample the MES at two times this frequency, i.e. 1kHz.

Consider a case with six raw MES as control input, and a data frame size of 8 bytes for each transmission. This gives the minimum bandwidth for the bus $B_{min} = 1000 * 6 * 8 * 8 = 384kbps$ As is stated in the functional requirements specification in appendix.A the current maximum bus traffic shall require only 40% of the bus bandwidth, in order to leave room for increased traffic. This means that the real bandwidth required is $B_{req} = B_{min}/0.4 = 768kbps$.

In the example above the data frame size of CAN was used, so for protocols with less overhead the required bandwidth will be less. SPI for example only sends one data byte per frame. The opposite is of course true for Ethernet and other higher level protocols, which have more overhead. Also it should be noted that more than one of the MES are likely to have the same recipient. This gives the opportunity to send several MES in the same message, drastically reducing the bandwidth required.

3.2.2 External Electronic Components

Since size and weight are critical in a prosthesis system, it is important that the bus standard of choice requires as few electronic components as possible. Ideally bus controllers should be integrated in the microprocessor.

3.2.3 Wiring

The functional requirements specification allows a maximum of two wires for data transfer(in addition to ground and power supply). This requirement is indeed rather strict, and by itself it rules out a few protocols. With an increased number of wires comes increased weight and maintenance problems, so it is probably a reasonable requirement despite its strictness.

3.2.4 Noise Sensitivity Reduction

As a prosthesis should be able to operate under most conditions, it is likely that SCIP will have to cope with noisy environments. This means that some sort of noise sensitivity reduction should be present. Two methods for realising this are:

Error reduction The first is implemented in the physical layer of the ISO/OSI model, and focuses on reducing the fault-rate, i.e. the percentage of corrupted frames. This could be achieved by sending the data over a differential bus. The voltage induced in both lines by electromagnetic noise will then be almost identical, and the difference between the voltage of the lines will be almost unaltered.

Error recovery The second method is implemented in the data-link layer and is discussed below in chap.3.2.9.

3.2.5 Bus Control

It is stated in the requirements specification that SCIP shall allow any two nodes to exchange data. It is therefore required that any node could initiate communication, a so called multi master solution.

3.2.6 Overhead

Frame overhead One aspect of overhead is the number of bits sent in each frame, which do not contain actual data, called frame overhead in this text. This includes start/stop conditions, error detection, adress-bits etc. Frame overhead should be kept small, but some error detection might be required.

Program overhead A second aspect of overhead is program overhead. This refers to the fact that more complex bus protocols require more program memory for the bus drivers. Program overhead could be reduced by writing specialized drivers. This however means work overhead, so ideally the bus protocol should not specify more features than needed.

3.2.7 Broadcasting

It is required that the bus supports broadcasting of messages. Broadcasting will greatly reduce overhead if there is one central node directing the traffic on the bus. This is likely to be the case for applications based on SCIP, and thus this is a reasonable requirement.

3.2.8 Message Priority

The only case where this is implemented in one of the lower layers of the ISO/OSI model, is when several nodes could send simultaneously. Arbitration on message means that if two nodes initiate a data transfer at the same time, the node with the lowest priority message will release the bus. In CAN for example a zero bit is dominant. Thus if one node outputs a 1 on the bus but discovers a 0 it immediately stops its transfer, knowing that another node is sending a higher priority message.

3.2.9 Error Detection

There are different ways of realising error discovery/recovery, but all have that incommon that some redundant data has to be added to the frame. The simplest error discovery method is parity bit stuffing, which means that one bit is added to make the data word contain either an odd or an even number of bits. This method can only discover an odd number of errors.
Cyclic redundancy check(CRC) is a more complex error detection algorithm. The details of how CRC works are beyond the scope of this text, but it applies

polynomial division to calculate a checksum which is used by the receiver to discover errors.

Message loss If the message was never sent then the algorithms mentioned above will not be of any help. What is often added to discover such errors is an acknowledge bit which a receiving node has to set either high or low. Further, the acknowledge bit is necessary for asking a sender to retransmit a corrupted frame. Alternatively a whole acknowledge frame might be sent.

Collision handling If two nodes start transferring data simultaneously, this should be detected by the bus, and some action should be taken to correct the situation. Otherwise the data will naturally be corrupted.

3.3 Comparison

The relevant functional requirements were specified above, and in some cases rewritten in a more concrete form. Table.1 summarizes how the different bus standards handle these requirements.

	UART	SPI	I2C	OneWire	CAN	USB	FireWire	Ethernet
Data wires	2	4	2	1	2	2	4	4
Bandwidth(bps)	2.7M	3M	400k	15k	1M	480M	400M	>1G
Noise sensitivity reduction	No	No	No	No	DB	DB	DB	DB
Bus control	1 to 1	SMMS	MM	SMMS	MM	SMMS	MM	MM
Frame overhead	3bit	0bit	12bit	–	47bit	80bit	128bit	512bit
Broadcasting	–	No	Lim	Lim	Yes	USB	Yes	Yes
Message priority	No	No	No	No	Arbitration	No	No	No
Error detection	Parity bit	No	No	CRC	CRC	CRC	CRC	CRC
Acknowledge bit	No	No	Yes	No	Yes	Ack frame	Ack Frame	Ack frame
Collision handling	No	–	CD	–	CD	–	–	CD

Table 1: Properties of different bus protocols

DB = Differential Bus
SMMS = Single master multiple slaves
MM = Multi master
CD = Collision detection(see chap.2.1.2)

3.4 Conclusion

Based on the information given in table.1, the CAN bus standard ISO 11898 is chosen to provide SCIP with physical and data link layer. The reasons for not choosing the other standards are summarized below.

- **UART**
  - Only suited for point to point communication.
- **SPI**
  - Too many wires.
  - No error detection.
  - Single master.
- **I2C**
  - Limited broadcast
  - No error detection.
  - No message priority.
- **OneWire**
  - Too low bandwidth
  - Limited broadcast
  - No message priority.
- **USB**
  - Increased message overhead compared to CAN.
  - Tiered star topology, resulting in the need for hubs if the wire requirement shall still hold.
  - Single master
- **TTP**
  - Decreased flexibility compared to CAN, since the sending cycle has to be reprogrammed when nodes are added/removed. Also the sending cycle should be reprogrammed when the control mode is changed.
  - More external components, as each node requires a bus guardian.
  - No message priority. In fact all messages will be delayed until the sender is allowed to access the bus.
- **FireWire**
  - Too much frame overhead
  - Too many wires
- **Ethernet**
  - Far too much frame overhead
  - Too many wires

4 Technical requirements specification

The main item of this section is a table which lists the technical requirements for SCIP. They were derived from the functional requirements specification as a part of the design process. The technical requirements specification covers all points of the functional requirements specification, with the exception of FR-08-04, FR-07 and FR-01. After discussion with my advisor Øyvind Stavdahl, FR-08-04, which states that the service controller should be able to download new firmware to other controllers on the bus, was found unrealistic. FR-07 which requires devices to be interchangeable was found to require far too much overhead. Further FR-01 which says that the bus shall have no more than two wires in total[3] was modified to no more than two data wires.

4.1 Technical requirements specification

The terms shall and should refer to functions that must be provided, and that may be left out if justified respectively.

Table 2: Technical requirements

Req.no	Requirement	Functional spec references
TR-01	The physical layer and the data link layer of SCIP will follow the definition for CAN stated in ISO 11898.	FR-01, FR-02, FR-04-01-01, FR-04-02, FR-04-03, FR-08-01, FR-11, FR-11-01, FR-11-02
TR-02	There shall be a node in the network that serves as bus controller(BC).	FR-08-02, FR-09
TR-02-01	The BC shall continually gather information about past errors, node resets etc.	FR-08-02
TR-02-02	The BC shall implement a watchdog timer which resets the BC in case of an error.	FR-09
TR-02-02-01	If the BC has been reset two times another watchdog timer overflow shall result in a shutdown of the whole system after it has entered a safe state.	FR-09-01
Continued...		

[3]That means both data and power on maximum two lines, and makes OneWire or wireless the only options possible.

Table 2: Technical requirements continued

Req.no	Requirement	Functional spec references
TR-02-02-02	Safe states have to be defined for every node in the network. *Comment: A safe state might be defined to be a state where a node gives no output to motors. An exception to this is a hand prosthesis, where a safe state should first open the hand if it is closed.*	FR-09-03
TR-02-03	The bus controller shall broadcast heartbeat messages with at least 10 Hz. If a node has not received these signals within 0.3 sec it shall assume a safe state and then reset. *Comment: The heartbeat message is used to tell the other nodes that the BC is operating properly, and that the bus connection is ok.*	FR-09-02
TR-02-03-01	All signals which are broadcast from the BC count as heartbeat messages.	FR-05
TR-02-04	The Device Controllers(DCs) and the Input Controller(InC) shall send heartbeat messages with at least 5 Hz. If the BC has not received heartbeat messages from a node within 0.5 sec the BC shall send a state request to that node. If the request is not answered within 0.5 sec the BC shall force this node into a safe state and then reset it. This requirement is only valid for the Running state of the DCs.	FR-05
TR-02-04-01	For the DCs and the InC all messages sent to the BC count as heartbeat messages as long as the senders adress is included, or the message is unique for that node.	FR-09-02

Continued...

Table 2: Technical requirements continued

Req.no	Requirement	Functional spec references
TR-02-05	The bus controller shall keep track of which node that sent the last message. If one node sends more than a specified number of messages in a row, this shall be considered a babbling idiot error and the node shall be forced into a safe state and then reset. *Comment: This calls for the reset-message to have highest priority(since the priority of the message from the babbling fool must be lower). In addition a watchdog timer might be necessary on every node, in case it fails in a way that prevents it from reading data from the bus.*	FR-09-02
TR-02-06	An error that still remains after a node has been attempted reset(by the system) a specified number of times shall be considered unrecoverable. If an unrecoverable error occurs, the system shall shutdown after it has entered a safe state.	FR-09-01
TR-03	The adressing space shall be sufficient to host 4 DCs, 1 InC plus a broadcast adress. This means that 3 bit adresses will be used.	FR-05-03, FR-04
TR-03-01	All DCs and the InC shall have predefined adresses. For the DCs these adresses will be based on device type. *Comment: This is only possible because there are a small number of standard device types like hand and elbow in an upper limb prosthesis system. Hybrid or new device types can use the unused adresses. Predefined adresses are used because it greatly simplifies the initialization procedure.*	FR-05
TR-03-02	Combined devices like a hand-wrist proshesis shall appear as two standard devices when communicating with the InC, but only one when communicating with the BC. *Comment: The reason for this is that a combined device acts as two devices when it comes to control, but it will only have one physical connection to the bus.*	FR-05
	Continued...	

Table 2: Technical requirements continued

Req.no	Requirement	Functional spec references
TR-04	The most significant bits of the identifier field shall define the message type, or message group. The lowest numbers shall be used for the most time critical data, i.e. reset and shut down messages. *Comment: The reason for this is that CAN arbitration gives highest priority to the lowest identifier value.*	FR-05-02
TR-05	There shall be a special node called Input Controller(InC), which is responsible for gathering input data from the user. It is also responsible for passing this data on to the correct DC and its DoF over the bus. If centralized control is supported it is also provided by the InC. *Comment: The reason for having centralized control and input management on the same node is to reduce traffic on the main bus. If there was a separate control node it would have to receive input data from the InC over the bus.*	FR-03
TR-05-01	The DCs shall be able to send status messages for all of its DoFs to the InC when this is requested. *Comment: These messages are only necessary during centralized control, and will then serve as feedback for the InC regulator.*	FR-03
TR-05-02	Control data of up to 16 bit resolution shall be supported.	FR-03-01
TR-06	There shall be a standardized initialization procedure that every DC has to follow at start-up, and after it has been reset.	FR-06, FR-07-01, FR-10
TR-06-01	Each DC and the InC shall, upon entering the network, deliver an initialization message to the BC. This message shall be retransmitted until an acknowledge message has been received from the BC. This initialization message shall contain the senders adress.	FR-06, FR-07-01
Continued...		

Table 2: Technical requirements continued

Req.no	Requirement	Functional spec references
TR-06-02	Each DC shall, upon entering the network, deliver an initialization message to the InC. This message shall be retransmitted until an acknowledge message has been received from the InC. This initialization message shall contain the senders adress, supported control signal types, supported data types and information about what degrees of freedom(DoF) are supported.	FR-03, FR-06, FR-07-01
TR-06-03	Each DC shall, after delivering the two messages above, deliver an initialization message to the InC for each DoF supported. These messages shall be acknowledged by the InC. These initialization messages shall contain the senders adress, supported control schemes, supported control strategies, supported control variables and extreme angles for each DoF.	FR-03, FR-07, FR-07-01
TR-07	A special node called the Service Controller(SC), shall provide a link between an external computer and the bus. This node shall only be attached during configuration and maintenance.	FR-08
TR-07-01	Configuration requires all DCs and the InC to be in the configure state. This state shall only be exited when the SC sends a message telling that the configuration is complete. *Comment: The point of the configure state is that the nodes will not interfere with the configuration bus traffic by sending hertbeat messages etc.*	FR-05
TR-07-02	The SC shall be able to send configuration messages to the InC about input signal types for each port, use of input signals and mapping of input signals to nodes. These messages shall be acknowledged by the InC.	FR-08-05, FR-08-05-01
TR-07-02-01	The SC shall be able to send configuration messages to the InC and the DCs about input signal types, control schemes, control strategies, control data types, control variables and mapping of input signals to nodes. These messages shall be acknowledged by the InC and the DCs	FR-08-05, FR-08-05-01
TR-07-02-02	The SC shall be able to send a special message to tell the nodes when the configuration is complete.	Follows from TR-07-01
	Continued...	

Table 2: Technical requirements continued

Req.no	Requirement	Functional spec references
TR-07-03	Upon a request from the SC, the BC shall supply information about past errors, node resets etc.	FR-08-02
TR-07-03-01	The BC shall at least store information about resets of itself in flash or SRAM. Other diagnostic information should also be stored in the same manner. *Comment: This is necessary in order to find the error when the BC itself causes it.*	Follows from TR-07-03
TR-07-04	Upon a request from the SC, the InC shall supply information about all prosthesis devices in the network. This information shall include device adress, supported control schemes, supported input types and supported degrees of freedom.	FR-08-05-01
TR-07-04-01	The information mentioned in TR-07-04 will be sent in several messages, and a special message shall therefore be sent to tell the SC when all messages are sent.	Follows from TR-07-04
TR-07-04-02	Upon a request from the SC, the InC shall also supply information about all of its input ports. This information shall include port number and input types supported for each port.	FR-08-05-01
TR-07-04-03	The InC shall store all information mentioned in TR-07-04 and TR-07-02 in flash or SRAM. *Comment: This is necessary in case the InC is reset.*	FR-08-05-01
TR-08	Each node except for the SC shall have a defined sleep state. This state shall be entered when no input has been received from the user for a specified time. The sleep state shall be exited when some user input changes.	FR-05
TR-08-01	DCs and the InC shall enter sleep mode upon a sleep command from the BC. They shall also be able to request permission to enter sleep mode.	Follows from TR-08
TR-08-01-01	DCs and the InC shall exit sleep mode upon a wake up command from the BC. They shall also be able to request permission to wake up.	Follows from TR-08
TR-08-02	The InC shall request to enter sleep mode as a result of inactivity from the user for a predefined time.	Follows from TR-08
TR-08-02-01	When in sleep mode the InC shall only transmit heartbeat messages.	FR-05

Continued...

Table 2: Technical requirements continued

Req.no	Requirement	Functional spec references
TR-08-02-02	The InC shall request to exit sleep mode as a result of change in user input.	Follows from TR-08
TR-08-03	The BC shall enter sleep mode when all other nodes are in sleep mode.	Follows from TR-08
TR-08-03-01	The BC shall exit sleep mode upon a wake up request from the InC.	Follows from TR-08
TR-09	The InC and the DCs shall always inform the BC when they enter a new state. *Comment: For more information about states see chap.5.1.*	Follows from TR-02-01
The End		

5 Design

Section 5.1 will start with a description of the different units in the network and their state transition diagrams. Then 5.2 gives a definition of the messages necessary for their interaction on the bus. In 5.3 the use of the CAN frame is specified for each message type. Following that is a declaration of all constants in section 5.4, and finally section 5.5 shows UML-diagrams for the two most complicated message sequences.

There are several open protocols available that are based on the CAN standard. Examples of such are DeviceNet, Smart Distributed System(SDS), CanOpen and M3S. The first three of these are very general, and a possibility would have been to let SCIP be based on one of those. This would have saved a lot of effort, but in being general those protocols also implement a lot of unneccesary functionality. This kind of program overhead should definitely be avoided in a specialized protocol like SCIP.

The aim was therefore to follow the example of M3S which is a protocol tailored for use in wheel chairs, but create a protocol tailored for use in prostheses instead. However elements from the protocols mentioned above have inspired some of the design choices in this report.

5.1 State transitions

SCIP separates between four kinds of units, the Bus Controller(BC), the Input Controller(InC), the Service controller(SC) and the Device Controllers(DCs). Their different tasks are described in chap.4. The SC is an external device, and will only be plugged into the bus occasionaly for configuration or checking the system status. For this reason it will be assumed that it is always in the running state, and thus no State Transition Diagram(STD) is necessary to explain its behaviour. For the BC, the InC and the DCs which are constantly in the network, STD:s will help in clarifying their behaviour.

5.1.1 STD for DCs

The state of a DC is mainly decided by the SC, BC or InC, as can be seen in fig.8. The node responsible for each state transition is written in paranthesis after the name of the trigger event. No name in paranthesis means that the node itself generates the event.

5.1.2 STD for the Bus Controller

The STD for the Bus Controller is shown in fig.9. Timer1 refers to a watchdog timer which should be reset periodically during normal operation. A failure to do so indicates an error in the BC software, and shall result in a reset(TR-02-02), or a shutdown(TR-02-02-01) depending on how many times the BC has been reset. Timer2 makes sure that the BC allows the other nodes a little time to enter safe states.

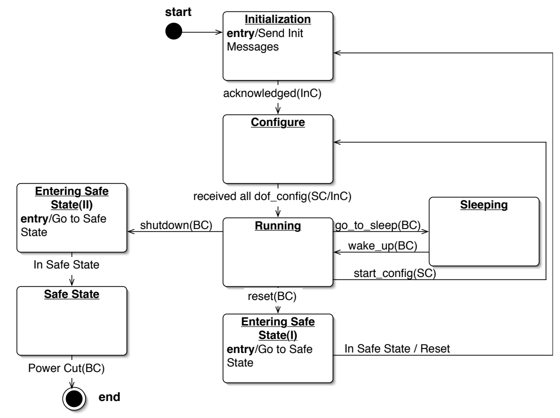

Figure 8: STD for a Device Controller

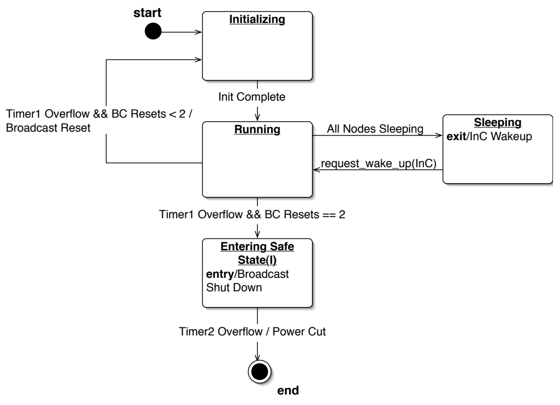
start
Initializing
Init Complete
Timer1 Overflow && BC Resets < 2 / Broadcast Reset
Running
All Nodes Sleeping
Sleeping
exit/InC Wakeup
request_wake_up(InC)
Timer1 Overflow && BC Resets == 2
Entering Safe State(I)
entry/Broadcast Shut Down
Timer2 Overflow / Power Cut
end

Figure 9: STD for the bus controller

5.1.3 STD for the Input Controller

In this STD(fig.9) it is noteworthy that even though the BC decides when the InC goes to sleep and wakes up, the actual event triggering this comes from the user. All user input is handled by the InC. This means that when the InC registers prolonged inactivity from the user it has to ask the BC to be sent to sleep, and when the InC again registers activity it has to ask to be woken. The reason for this is that the BC shall control main events on the bus during regular operation. This includes state transitions for the other nodes.

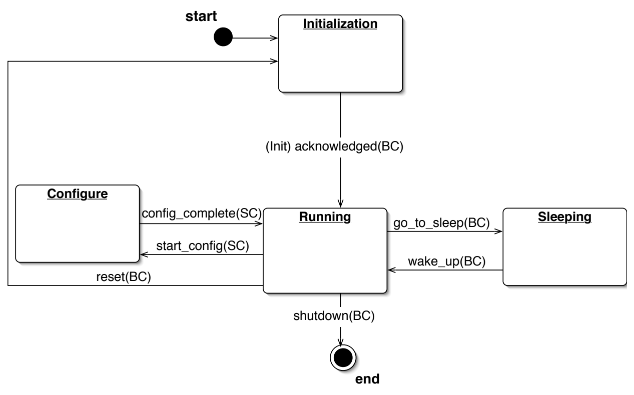

Figure 10: STD for the Input Controller

5.2 Message Derivation

In this section the messages needed for implementing a version of SCIP based on the Technical Requirements Specification are derived.

5.2.1 Initialization Messages

During initialization the DCs and the InC shall identify themselves to the BC, and the DCs shall supply the InC with information about themselves. The BC and the InC shall acknowledge these messages upon reception. The following messages are necessary to achieve this:

Message Name	Derived from	Description
node_init	TR-06-01	Sent by the DCs and the InC to the BC, this message contains only the adress of the sender.
dc_init	TR-06-02	Sent by the DCs to the InC, these messages contain information about what data types, input types and degrees of freedom the DC supports.
dc_dof_init	TR-06-03	Sent by the DCs to the InC, these messages contain information about what control schemes, control strategies and control variables the given DoF supports. Also provides information about extreme angles.
acknowledge	TR-06-01, TR-06-02, TR-06-03	Used to confirm that a message has been received

Table 3: Initialization Messages

5.2.2 Control Messages

For control of a prosthetic device, the only information exchange needed are control data from the Input Controller to the Device Controllers, and status variables from the DCs to the InC. Control data is transmitted from the InC with a predefined frequency, whereas status variables are sent upon demand. The three new message types introduced are shown in table.4:

Message Name	Derived from	Description
control_data	TR-05	Sent by the InC to the DCs, this message contains control data for some of the nodes DoFs.
request_status_vars	TR-05-01	Sent by the InC to one of the DCs in order to ask for the measured values of the control variable.
status_vars	TR-05-01	Sent by the DCs to the InC, this message contains the measured values of the control variables for some of the nodes DoFs.

Table 4: Control Messages

5.2.3 Configuration Messages

When the SC is connected to the bus for the first time, the service person must configure the system. The necessary information exchange during configuration is listed in TR-07-02, TR-07-04 and TR-07-04-02 in chap.4. The configuration procedure will follow the sequence described below.

- The SC starts the configuration by sending a config message.
- The InC and the DCs react on the config message by entering the configure state.
- The BC tells the SC when all nodes are in the configure state.
- The InC sends information about its input ports to the SC. The SC acknowledges each of these messages upon reception.
- The InC supplies information about the DCs available, and their DoFs. The SC acknowledges each of these messages upon reception.
- When the InC has supplied the SC with all available information it sends a message to tell this.
- The service person sets up the system on an external computer.
- When the service person is done the SC tells the InC about these settings. The settings include input type used for each input port, input function, and mapping of each input to DC, DoF and rotation direction. Further it includes control scheme, control strategy, control data format, control variable and control signal type for each DC and its DoFs. The InC acknowledges each set up message upon reception.

- When all configuration is completed, the SC sends a special message which states this.

Message Name	**Derived from**	**Description**
start_config	TR-07-04, TR-07-04-02	Sent by the SC to start the configuration sequence.
ready_to_config	TR-07-04, TR-07-04-02	Sent by the BC to start the configuration sequence. In effect this message works as a request to the InC to return information about itself and the DCs.
inc_port_info	TR-07-04-02	This message is sent by the InC and supplies the SC with information about input port nr, and input types supported.
inc_dc_info	TR-07-04	This message is sent by the InC and supplies the SC with information about what data types and input types the DC supports.
inc_dof_info	TR-07-04	This message is sent by the InC and supplies the SC with information about what control schemes, control strategies and control variables the DoFs of all DCs support.
info_complete	TR-07-04-01	Sent by the InC to inform the SC that information about all DCs have been sent.
inc_port_config	TR-07-02	Sent by the SC to tell the InC how the input ports are configured. This includes what input type shall be used, the use of that input and mapping of the input to the correct control data(1 or 2) of the correct DoF of the correct DC.
dof_config	TR-07-02	Sent by the SC to tell the InC and the DCs how the DCs are configured. This message could also be sent by the InC to a DC that has been reset. It includes information about what control scheme, control strategy, control variable, data type and input type shall be used.
config_complete	TR-07-02-02	Sent by the SC to tell all nodes that the configuration procedure is completed.

Table 5: Configuration Messages

5.2.4 Housekeeping Messages

Housekeeping refers to all messages needed to check that the bus and the nodes on the bus are operating properly. It also includes messages to handle a situation where this is not the case. Finally messages that are involved in changing the mode of operation of nodes are included here. The following housekeeping messages can be derived from the technical specification:

Message Name	**Derived from**	**Description**
heartbeat	TR-02-03, TR-02-04	Sent periodically by all kinds of nodes except for the SC, to tell the other nodes that one is operating properly. Heartbeat signals are only necessary when no other signals have been sent for a given time.
request_state_info	TR-02-04	Used by the BC to receive information about the current state of the targeted DC or the InC
state_info	TR-09, TR-02-04	Sent by the DCs or the InC in answer to the request_state_info message, and when they change state.
reset	TR-02-02-01, TR-02-03, TR-02-05	Used by the BC to reset one ar all of the nodes in the network in the case of an error.
shutdown	TR-02-04, TR-02-06	Used by the BC to shut down all of the nodes in the network in the case of an unrecoverable error.
go_to_sleep	TR-08-01	Used by the BC to send one or all of the nodes to sleep.
wake_up	TR-08-01-01	Used by the BC to wake up one or all of the nodes.
request_sleep	TR-08-01, TR-08-02	Used by the InC or DCs to ask the BC to send it to sleep.
request_wake_up	TR-08-01-01, TR-08-02-02, TR-08-03-01	Used by the InC or DCs to ask the BC to wake it up.

Table 6: Housekeeping Messages

5.3 Message Layout

This chapter describes the use of the data- and ID-fields of the CAN frame. The ID-field has 11 bit and the data-field may contain up to 8 byte. This report is meant as a suggestion for the specification of SCIP, and it is likely that more messages will be added in time. Therefore it is important to leave room for growth when specifying the use of the ID-field. Also it is important not to let the CAN ID-field start with 7 or more consecutive recessive bits(1), since some CAN transceivers do not support this.

5.3.1 Use of the ID-field

In order to find a good way of utilizing the CAN ID-field, the messages defined above will be sorted with respect to structural similarities. The following main groups can be identified:

- **Adressed messages with no data**
  - acknowledge
  - node_init
  - reset
  - request_state_info
  - heartbeat
  - request_sleep
  - go_to_sleep
  - request_wake_up
  - wake_up
- **Non adressed messages with no data**
  - info_complete
  - ready_to_config
  - shutdown
  - start_config
  - config_complete
- **Hierarchically adressed messages with no data**
  - request_status_vars
- **Adressed messages containing data**
  - dc_init
  - inc_dc_info
  - state_info

- **Non adressed messages containing data**
  - inc_port_info
  - inc_port_config
- **Hierarchically adressed messages containing data**
  - control_data
  - status_vars
  - dc_dof_init
  - inc_dof_info
  - dof_config

Adressed messages refer to messages that contain an adress. This adress might specify sender, receiver or just the node that the information in the message is about. Hierarchically adressed means that one or several Degrees of Freedom(DoFs) within a DC are adressed. An example of this is when the InC sends a *request_status_vars* to the flexion/extension DoF of a wrist device.

The number of messages specified above is 25, and thus 5 bit should be sufficient for identifying message type. Further it is stated in the technical requirements specification that adresses shall be 3 bit. Fig.11 shows how the different kinds of message groups listed above utilize the ID-field of the CAN frame. Note that the message *state_info* uses the last three bit of the ID-field to send data about its current state. Another thing worth commenting on is that group 4 only contains two messages, both concerned with the input ports of the InC. This allows the port nr to be specified in the ID-field. The message nr part of the CAN ID-field for message group 7 is explained in the next section.

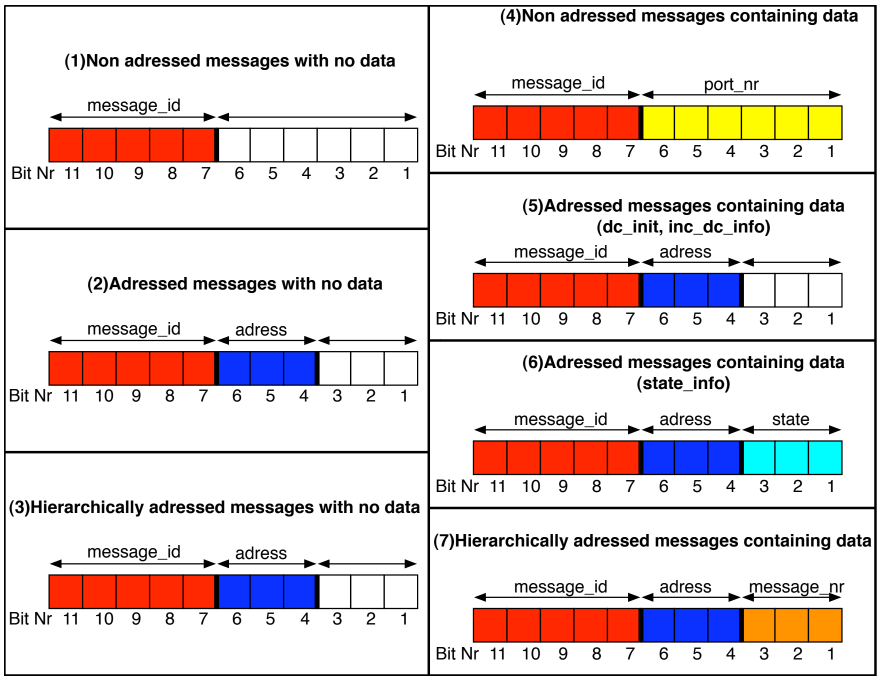
(1)Non adressed messages with no data
message_id
Bit Nr 11 10 9 8 7 6 5 4 3 2 1
(2)Adressed messages with no data
message_id
adress
Bit Nr 11 10 9 8 7 6 5 4 3 2 1
(3)Hierarchically adressed messages with no data
message_id
adress
Bit Nr 11 10 9 8 7 6 5 4 3 2 1
(4)Non adressed messages containing data
message_id
port_nr
Bit Nr 11 10 9 8 7 6 5 4 3 2 1
(5)Adressed messages containing data (dc_init, inc_dc_info)
message_id
adress
Bit Nr 11 10 9 8 7 6 5 4 3 2 1
(6)Adressed messages containing data (state_info)
message_id
adress
state
Bit Nr 11 10 9 8 7 6 5 4 3 2 1
(7)Hierarchically adressed messages containing data
message_id
adress
message_nr
Bit Nr 11 10 9 8 7 6 5 4 3 2 1

Figure 11: Use of the CAN ID-field

5.3.2 Use of the Data-field

For the messages in the right part of fig.11, the Data-field will contain different kinds of information. An exception to this is the *state_info* message which includes its data in the ID-field. Type 3 messages utilize the Data-field to adress DoFs within a DC. It will therefore be necessary to specify the use of the Data-field for each kind of message in groups 3, 4, 5 and 7.

request_status_vars is the only message type in group 3. It will use 1-4 bytes of data to specify which DoFs it requests status from. The first byte corresponds to DoFs 0-7, the second 8-15 and so on. A 1-bit stands for request while a 0-bit stands for no request. In the example of fig.12 DoF nr 1,4,13 and 21 should respond with their status variables. The DoFs of the standard prosthesis devices will all be given a specific number in chap.5.4

Group 4 consists of *inc_port_info* and *inc_port_config*. *inc_port_info* contains one byte of data, which tells what input types the port supports. The port nr was given in the ID-field. The different input types are specified in chap. 5.4.

inc_port_config contains two bytes of data. The first three bit of the first byte states what input type the port will receive, the fourth bit states whether it

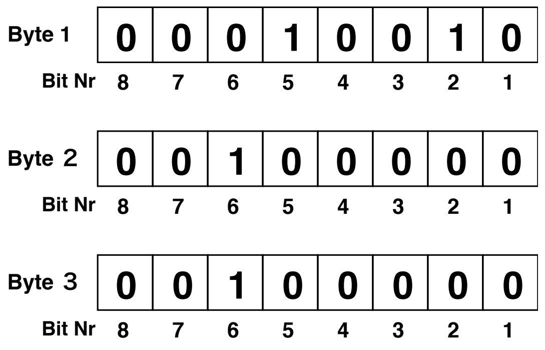

Figure 12: Example of use of the Data-field for *request_status_vars*

is the first or the second of the control input[4] and the next three bit specify the use of the input. The second byte is also divided in two parts. The first part is three bit and gives the adress of the DC, that shall receive the input. The second unit is five bit and determines which DoF within the DC that is adressed.

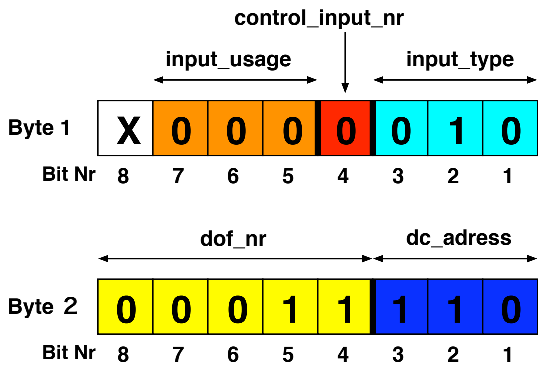

Figure 13: Example of use of the Data-field for *inc_port_config*

The example given in fig.13 shows how the data-field would look when a port was configured for supplying DoF nr 3 of DC nr 6 with the first control input of type 2. Input usage is only set if the adress is set to broadcast. dc_adress, input_type and input_usage will be specified in chap.5.4 .

Group 5 has two specified messages which are *dc_init* and *inc_dc_info*. The *dc_init* message has 2-5 bytes in the data-field, where the first byte specify what

[4]I.e. positive or negative rotation direction

data types the DC supports. A one bit means that the data type with number corresponding to the bit nr is supported. Data types are bit, byte, 16bit etc, and their number mapping is specified in chap.5.4 . The next byte specifies what input types the DC supports in an identical manner. The last 1-4 bytes contain supported DoFs in a manner identical to that for *request_status_vars*. This is all illustrated in fig.14, where DoFs 2 and 10[5] are at least supported(the last two byte are not shown in the figure). Also data types 1 and 6, and input types 1 and 4 are supported.

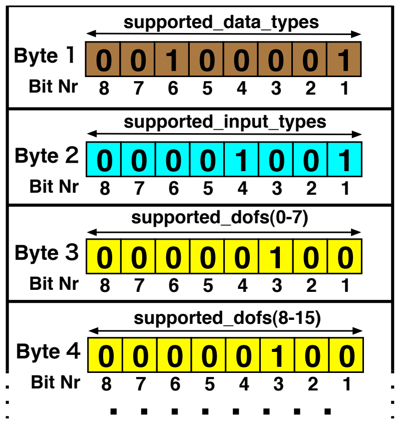

Figure 14: Example of use of the Data-field for *dc_init*

The message type *inc_dc_info* uses the data field in a manner identical to *dc_init* except for that it lacks the last 1-4 bytes about the DoFs.

Group 7 is the largest group and contains the message types; *control_data, status_vars, dc_dof_init, inc_dof_info* and *dof_config*. These message types utilize the data-field in different ways, so each has to be specified separately. The last three messages have some data incommon.

When it comes to *control_data* one message is always adressed to a single DC. The adress of the DC is specified in the ID-field as was shown in fig.11 above. On the other hand one message might contain one or two control signals to one or several DoFs within the DC. The adressing of DoFs is done by allowing the first data byte and the three LSB of the ID-field(message_nr) to tell which degrees of freedom will receive data in this message. The first data byte also tells whether the message contains the first, the second or both control signals for each of the DoFs. There is support for up to 32 DoFs for each DC. The adressing scheme is illustrated in fig.15. As can be seen there is room for 4 DoFs in each message, and the number of the first DoF in one message can therefore be found as $message\_nr * 4$.

[5]Remember that the first DoF number is 0.

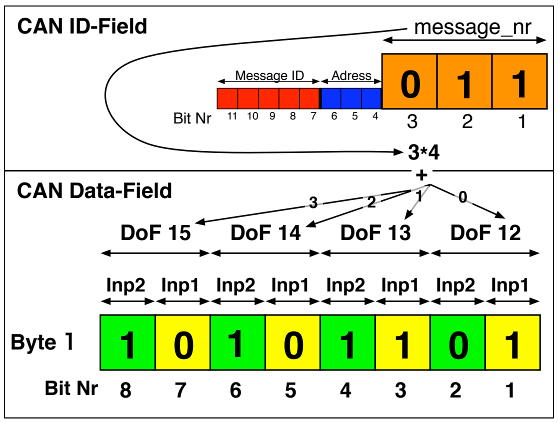

Figure 15: Use of the Data-field and ID-field for adressing *control_data*

In the example of fig.15 DoF 12 receives control input 1, DoF 13 both 1 and 2, and DoFs 14 and 15 receive control input 2. These data are contained within data bytes 2-8. The data is structured according to the following rules:

- All data are sorted on DoF in rising order, and thereafter on input nr in rising order.
- If any data are of the type bit, these data are all stored in the second data byte.
- Data of 9-16 bit is stored in two consecutive bytes with the LSB in the first byte.

If both input for DoF 13 was of the type byte, the input for DoF 14 was of the type 16bit, and the input for DoFs 12 and 15 were single bit this would look like in fig.16.

The next message type is *status_vars*. The status for each degree of freedom can be either the deviation angle or the rotation speed. In addition a third and fourth value is left to be specified. The deviation angle is given from the standard anatomical position mentioned in chap.2.2.1. It shall always be 9 bit allowing the angle to be expressed in degrees, ranging from 0 to 360. The rotation speed is given in degrees/s and specifies the speed of the end device, not the motor. It is given as a single byte.

The DoFs identify themselves in a similar manner as they were adressed in the *control_data* message described above. The only difference is that each message number correspond to eight different DoFs instead of four. This is illustrated in fig.17.

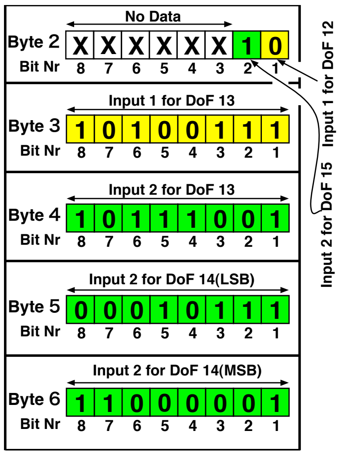

Figure 16: Use of the Data-field for sending *control_data*

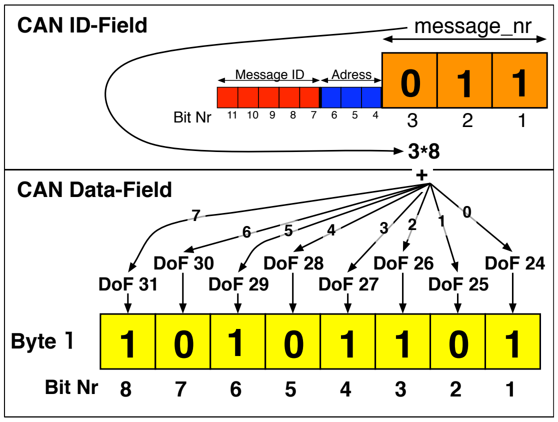

Figure 17: Use of the Data-field and ID-field for adressing *status_vars*

Fig.18 shows an example where DoFs 8 and 14 send their status variables. In the example the variables are assumed to be the deviation angles as can be seen by the fact that the data consists of 9 bit. It is also illustrated that the MSB of all 9 bit data words are sent in data byte 2. This is always the case. The different status types are declared in chap.5.4.

For the message types *dc_dof_init*, *inc_dof_info* and *dof_config* each message contain information about one single DoF. They are all identified or adressed in an identical manner as the last message type.

These four message types will always have the same payload, making things a bit easier than for the other messages of group 7. For *dc_dof_init* the data consists of the following parts: supported_control_schemes, supported_control_strategies, supported_control_variables, and extreme_angles. These data are laid out in the

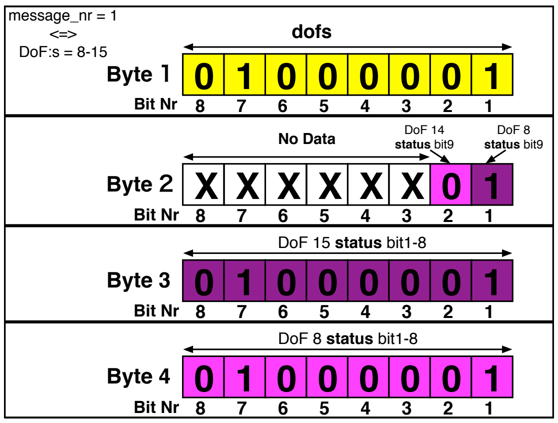

Figure 18: Use of the Data-field for sending *status_vars*

data field in accordance with fig.19. In this example DoF nr 2 is initialized(given that message_nr is 0) and it supports control strategies 3 and 7, control scheme 1, control variables 0,1 and 3 and has a maximum angle of 65(64+1)and a minimum angle of 263(256+4+2+1). Note that the minimum angle is larger than the maximum angle due to the fact that negative angles are recalculated by adding 360∠.

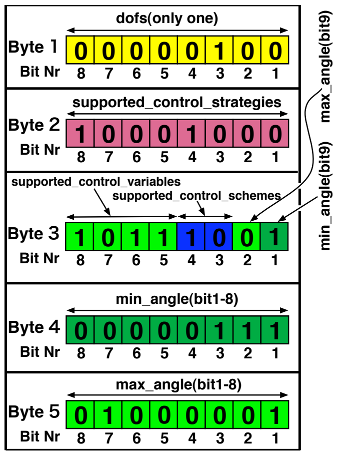

Figure 19: Use of the Data-field for *dc_dof_init*

inc_dof_info passes some of the information in the *dc_dof_init* messages on to the service controller. Its use of the data-field will be identical to *dc_dof_init*, except for the 18 bit concerning max and min angles.

Finally there is the message type *dof_config*. It shall provide the variables control_scheme, control_strategy, control_variable, data_type and input_type. The data-field usage is shown in fig.20. In this example DoF nr 2 is configured(given that message_nr is 0). It shall use control strategy 6 and control scheme 0. The variable regulated is of type 1, the data is of type 2 and the input is of type 4.

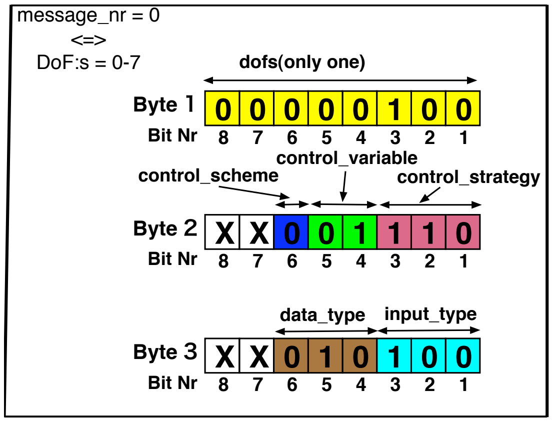

Figure 20: Use of the Data-field for *dof_config*

5.3.3 Summary

All message types are summarized in table.7. This table gives message type, contents of the ID- and data-field, sender, receiver and state of the sender.

Table 7: Summary of messages

Message name	message _id	Send-er	Re-ceiver	State of Sender	Contents of ID-field	Contents of Data-field(bit)
acknowledge	10000	All	All	INIT, CONFIG, RUN-NING	message_id, adress	–
node_init	10011	DC, InC	BC	INIT	message_id, adress	–
reset	00001	BC	DCs, InC	RUNNING	message_id, adress	–
request_state_info	00101	BC	InC, DCs	RUNNING	message_id, adress	–
heartbeat	00100	DCs, InC, SC	BC	RUNNING	message_id, adress	–
request_sleep	01100	DCs, InC	BC	RUNNING	message_id, adress	–
go_to_sleep	01011	BC	DCs, InC	RUNNING	message_id, adress	–
Continued...						

Table 7: Summary of messages continued

Message name	message_id	Sender	Receiver	State of Sender	Contents of ID-field	Contents of Data-field(bit)
request_wake_up	01010	DCs, InC	BC	SLEEP	message_id, adress	–
wake_up	00111	BC	DCs, InC	RUNNING	message_id, adress	–
info_complete	11110	InC	SC	CONFIG	message_id	–
ready_to_config	11011	BC	SC	RUNNING	message_id	–
shutdown	00000	BC	DCs, InC	RUNNING	message_id	–
start_config	11010	SC	DCs, InC	RUNNING	message_id	–
config_complete	11001	SC	DCs, InC	RUNNING	message_id	–
request_status_vars	01110	InC	DCs	RUNNING	message_id, adress,	dofs(8-32)
dc_init	10010	DCs	InC	INIT	message_id, adress	supported_data_types(8), supported_input_types(8), dofs(8-32)
inc_dc_info	11000	InC	SC, InC	CONFIG	message_id, adress	supported_data_types(8), supported_input_types(8)
state_info	00110	DCs, InC	BC	Any State	message_id, adress, state	–
inc_port_info	10111	InC	SC	CONFIG	message_id, port_nr	supported_input_types(8)
inc_port_config	10110	SC	InC	RUNNING	message_id, port_nr	input_type(3), control_input_nr(1), input_usage(3), adress(3), dof_nr(5)
control_data	01111	InC	DCs	RUNNING	message_id, adress, message_nr	dofs(8), control_data(1-56)
status_vars	01101	DCs	InC	RUNNING	message_id, adress, message_nr	dofs(8), status_vars(8-56)
dc_dof_init	10001	DC	InC	INIT	message_id, adress, message_nr	dofs, supported_control_strategies(8), supported_control_schemes(2), supported_control_variables(4), max_angle(9), min_angle(9)
Continued...						

Table 7: Summary of messages continued

Message name	message_id	Sender	Receiver	State of Sender	Contents of ID-field	Contents of Data-field(bit)
inc_dof_info	10101	InC	SC	CONFIG	message_id, adress, message_nr	dofs(8), supported_control_strategies(8), supported_control_schemes(2), supported_control_variables(4)
dof_config	10100	SC	DCs, InC	RUNNING	message_id, adress, message_nr	dofs(8), control_strategy(3), control_variable(2), control_scheme(1), input_type(3), data_type(3)
The End						

5.4 Declaration of Constants

In this section all known constants are given specific values. The range of these values depend on what variable the constant is tied to.

5.4.1 message_id

All message types are given message_id values in table.7. The message_id gives the message its priority on the bus. The higher the priority of the message, the lower must the message_id be. *reset* and *shutdown* are given highest priority. Other housekeeping messages are also given high priority, as the functionality of the bus depends on them. Messages sent during configuration are given lowest priority, as they are sent very rarely. Initialization messages are also given low priority, for the same reason. When all message types have been given message_id the following values are left unspecified:

- 00010
- 00011
- 01000
- 01001
- 11100
- 11101
- 11111

This leaves two free high-priority messages, two medium-priority and two low-priority messages for future use. The value 11111 must not be used[6].

[6]This is done to avoid the risk of having a message with 7 consecutive 1 bits as some CAN transceivers do not handle this.

5.4.2 control_strategy

The unspecified values for the variable control_strategy are 3-7.

Constant Name	**Binary Value**	**Decimal Value**
SINGLE_SITE	000	0
TWO_SITE	001	1
CENTRAL_CONTROL	010	2

5.4.3 control_scheme

The unspecified values for the variable control_scheme are none.

Constant Name	**Binary Value**	**Decimal Value**
PROPORTIONAL	0	0
ON_OFF	1	1

5.4.4 control_variable

The unspecified values for the variable control_variable are 2-3.

Constant Name	**Binary Value**	**Decimal Value**
POSITION	00	0
VELOCITY	01	1

5.4.5 input_type

The unspecified values for the variable input_type are 4-7. See chap.2.3 for a review of the different sensor input.

Constant Name	**Binary Value**	**Decimal Value**
MES	000	0
FSR	001	1
SWITCH	010	2
LINEAR_POTENTIOMETER	011	3

5.4.6 input_usage

The unspecified values for the variable input_usage are 3-7.

Constant Name	**Binary Value**	**Decimal Value**
CONTROL	000	0
SWITCH_MOTOR_DIR	001	1
CHANGE_CONTROL_SCHEME	010	2

5.4.7 data_type

The unspecified values for the variable data_type are 4-7. The reason for predefining 10 bit and 16 bit is that these are very common resolutions for A/D converters. Byte is very convenient for sending control data that does not need higher resolution, and bit are used by switches.

Constant Name	**Binary Value**	**Decimal Value**
BIT	000	0
BYTE	001	1
10_BIT	010	2
16_BIT	011	3

5.4.8 adress

This variable refers to the adress of a node on the bus. The unspecified values for the variable adress are 6-7.

Constant Name	**Binary Value**	**Decimal Value**
BROADCAST	000	0
INPUT_CONTROLLER	001	1
HAND_DEVICE	010	2
WRIST_DEVICE	011	3
ELBOW_DEVICE	100	4
SHOULDER_DEVICE	101	5

5.4.9 state

This variable refers to the current state of a node. The unspecified values for the variable state is 7.

Constant Name	**Binary Value**	**Decimal Value**
INIT	000	0
CONFIG	001	1
RUNNING	010	2
ENTERING_SAFE_1	011	3
ENTERING_SAFE_2	100	4
SAFE	101	5
SLEEP	110	6

5.4.10 Shoulder dofs(Degrees of Freedom)

The unspecified values for the variable dofs for the shoulder are 3-31.

Constant Name	**Binary Value**	**Decimal Value**
FLEXION_EXTENSION	00000	0
ABDUCTION_ADDUCTION	00001	1
PRONATION_SUPINATION	00010	2

5.4.11 Elbow dofs(Degrees of Freedom)

The unspecified values for the variable dofs for the elbow are 1-31.

Constant Name	**Binary Value**	**Decimal Value**
FLEXION_EXTENSION	00000	0

5.4.12 Wrist dofs(Degrees of Freedom)

The unspecified values for the variable dofs for the wrist are 3-31.

Constant Name	**Binary Value**	**Decimal Value**
FLEXION_EXTENSION	00000	0
ABDUCTION_ADDUCTION	00001	1
PRONATION_SUPINATION	00010	2

5.4.13 Hand dofs(Degrees of Freedom)

The unspecified values for the variable data_type are 21-31. The abbreviations used are flex_ext for flexion/extension, abd_add for abduction/adduction and pron_sup for pronation/supination. Further the number in each constant name stands for fingers. They are numbered from thumb to little finger in rising order, i.e. thumb = 1, index finger = 2 and so on. MCP, DIP, PIP, IP, and CMC are different joints of the hand. For a review of the joints of the human hand see chap.2.2.5.

Constant Name	**Binary Value**	**Decimal Value**
FLEX_EXT_CMC_1	00000	0
FLEX_EXT_MCP_2	00001	1
FLEX_EXT_MCP_3	00010	2
FLEX_EXT_MCP_4	00011	3
FLEX_EXT_MCP_5	00100	4
ABD_ADD_CMC_1	00101	5
ABD_ADD_MCP_2	00110	6
ABD_ADD_MCP_3	00111	7
ABD_ADD_MCP_4	01000	8
ABD_ADD_MCP_5	01001	9
FLEX_EXT_MCP_1	01010	10
FLEX_EXT_PIP_2	01011	11
FLEX_EXT_PIP_3	01100	12
FLEX_EXT_PIP_4	01101	13
FLEX_EXT_PIP_5	01110	14
FLEX_EXT_IP_1	01111	15
FLEX_EXT_DIP_2	10000	16
FLEX_EXT_DIP_3	10001	17
FLEX_EXT_DIP_4	10010	18
FLEX_EXT_DIP_5	10011	19
ABD_ADD_MCP_1	10100	20

5.5 Message Sequences

This chapter gives UML-diagrams for the message sequences for configuring the nodes, and for initializing the DCs and the InC.

5.5.1 Initialize Sequence

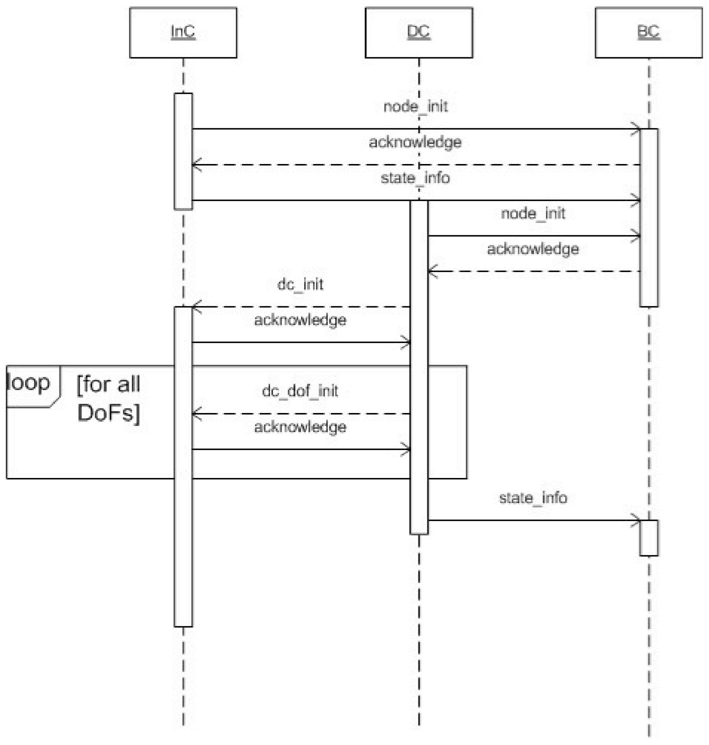

Figure 21: The message sequence for initialization of the DCs and the InC.

5.5.2 Configure Sequence

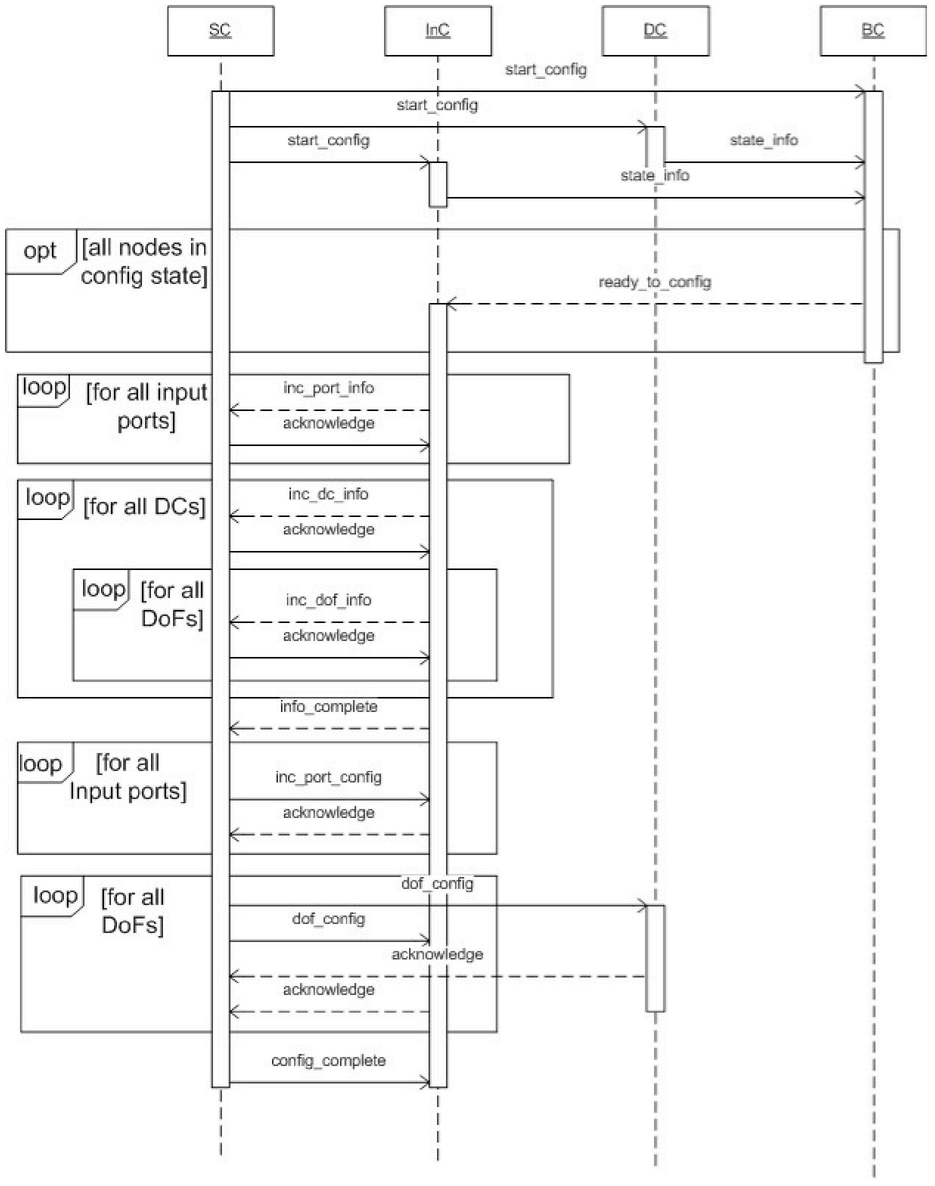

Figure 22: The message sequence for configuration of the system.

6 Discussion

6.1 Choice of Bus

It seems that choosing the CAN bus as base for SCIP was a good choice. It provides a lot of very useful functionality, like message priority which SCIP implements by letting message_id provide the MSB of the ID-field. When the protocol shall be tested it is also likely that the automatic retransmit upon error, combined with the CRC code, will make the protocol very noise tolerant. One small drawback with CAN is the fact that there is only room for eight data bytes per message. This is only a problem when sending control data to DCs with many DoFs and control data for several DoFs needs updating. In this case several messages might be required, creating some extra overhead. This might provide an argument to check other standards like TTP where the data-field is twice as big. It should be noted however that this situation will not arise frequently with todays prostheses.

6.2 Node Interaction

It was attempted to make the node interaction demand as little bus-traffic as possible, while still maintaining reasonable complexity. This is the reason why there is support for a sleep state, and the possibility of sending control data to several degrees of freedom simultaneously. It is possible that a more optimal solution could be found for utilizing the data field. For example if several 10 bit data shall be transfered, the two extra bit of each data might be sent in the same byte. The benefit of this depends on how often 10 bit data will be used compared to 8 bit data. The reason for not supporting that solution in this design was simplicity.

6.3 Use of the CAN ID-field

The number of messages that proved necessary for implementing SCIP turned out to be greater than first assumed. In fact only five bit-combinations remain free for use in message_id. I.e. only five more message types could be specified. This could easily turn out to be insufficient. A solution to this could be to utilize all the unspecified bits in message group 1(see fig.11). If these message_id fields were expanded with three bit we would get $8 * 4 = 32$ new message types.

7 Conclusion

This report should be a good base for developing the SCIP protocol further. The technical requirements specification cover all the functional requirements.[7] From the technical requirements specification, four different kinds of devices were derived. The bus controller, the input controller, the device controller and the service controller. Their actions and interactions[8] are described in such detail that it should be easy to implement a solution based upon this report. It is however highly likely that some necessary messages have been overlooked. The unused bit-combinations in the ID-field should prove sufficient to cover this.

The fact remains that some part of the protocol should have been implemented. The author choose to attempt to do a thorough design job rather than rushing into some half measure implementation. The result of this design job is a detailed technical requirements specification, specifications of the node types and their states and an overview of the necessary message types. Further, all constants have been given specific values and the most complicated message sequences have been described with UML-diagrams.

[7]With the exception of the ones that were found too strict

[8]In the form of message transfer

8 Future Work

8.1 Consistency Check

The design proposal laid out in this report aims to be no more than a proposal. This means that it should be thoroughly and critically read through and discussed. This should attempt to rule out lacks, inconsistencies and inefficient solutions. Adjustments and changes should be made where the need is found. It is not easy to be overly critical to ones own ideas so this part is important in order to achieve a good implementation.

8.2 Implementation

The next step would be to let someone implement a solution based on the revised design. As the protocol design is quite detailed this should be a rather easy task. The proposed programming language is C.

8.3 Testing

Once a testable version of SCIP exists, it should be downloaded to microcontrollers and thoroughly tested. The testing should verify whether the desired functionality works, and should also test the system under maximum bus traffic to make sure that it does not get overloaded.

8.4 Alternative Physical Layer Solutions

It was mentioned in chap.3.1 that existing wireless solutions are not suited for SCIP. It was also mentioned that Wibree might provide an interesting alternative once its specification is finished. The lack of need for wires for data transfer is a very desirable feature, so this should absolutely be investigated. If Wibree was chosen as the physical layer for SCIP, it is still likely that much of the specification in this report could be used.

References

Catsoulis, J. (2003), *Designing Embedded Systems*, 4 edn, Oreilly and Associates, Inc, 1005 Gravenstein Highway North, Sebastopol, CA.

Karnå, D. (2006), 'Design of pinocchio(prosthesis integrated node for communication control and hmi input/output)'.

Kopetz, H. (1998), A comparison of can and ttp, Technical report, Institut für Technische Informatik, Technische Universität Wien.

Muzumdar, A. (2004), *Powered Upper Limb Prostheses*, Springer-Verlag Berlin Heidelberg New York.

Olsen, O. A. (1998), *Instrumenteringsteknikk*, 5 edn, Tapir Forlag, Trondheim.

Onshus, T. (2006), *Instrumenteringssystemer*, 4 edn, Institutt for teknisk kybernetikk, NTNU - Norges teknisk-naturvitenskapelige universitet, Trondheim.

Pazul, K. (1999), Controller area network (can) basics, Technical report, Microchip Technology.

Rijpkema, H. & Girard, M. (1991), 'Computer animation of knowledge-based human grasping'.

Tanenbaum, A. S. (2003), *Computer Networks*, 4 edn, Pearson Education, Inc., Upper Saddle River, New Jersey.

Wickelgren, I. J. (1997), 'The facts about firewire'.

Øyvind Stavdahl (2002), *Optimal Wrist Prosthesis Kinematics*, Institutt for teknisk kybernetikk, NTNU - Norges teknisk-naturvitenskapelige universitet, Trondheim.

A Functional Requirements Specification

This is the functional requirements specification that served as base for this master thesis.

Functional Requirements Specification

This section presents a structured list of functional requirements that are derived from the meta-spec in the previous section. References are made to the meta-spec wherever applicable.

The terms **should** and **shall** refer to functions that are desirable that may be left out if justified and functions that must be provided, respectively.

Functional Requirements

Req. No.	Requirement	Spec or Meta-Spec References
FR-01	The bus **should** require a minimal number of electrical wires. The total number of wires (data and power) **shall** not exceed two. *Comment: Ideally data and power on the same lines, or wireless transmission (e.g. ZigBee)*	MS-05-01-02 MS-05-01-03 MS-05-01-05 MS-05-02-05 MS-05-03-02 (MS-07-01)
FR-02	The bus **should** require a minimal number of electronic components. *Comment: Desirable: all protocol logics integrated with a microcontroller*	MS-05-01-05 MS-05-03
FR-03	There **shall** be support for presently used control schemes. *Comment: Known schemes include sequential (uncoordinated) ON/OFF, proportional velocity or position control of a set of joints; simple coordination of several joint positions; hierarchic control (SAMS, SUVA), and more TBC.* Support for *does not imply that these are all implemented, but all the necessary data transmissions shold be possible within the protocol.* *Comment: The bus is **not** required to support present components (analogue EMG signals etc.), only the high-level control principles.*	MS-01 MS-02-01
FR-03-01	The protocol **shall** specify data formats with sufficient ranges and resolution for the transfer of control signals and state and status variables for the classical control schemes. *Comment: Cf. FR-03.*	MS-05-01-04-02
FR-04	The bus **shall** leave room for future growth. *Comment: Address space etc. not "fully specified", some left for future enhancements. Some bus bandwith surplus.*	MS-01 MS-05-03-03
FR-04-01	There **shall** be flexibility and vacant resources for accommodating more advanced schemes.	MS-01 MS-02-02 MS-02-03
FR-04-01-01	The specified bus traffic **shall** not exceed 40% of the bus bandwidth. *Comment: The* specified *traffic is the traffic that is planned to take pace during normal operation conditions with "traditional" control schemes, cf. FR-03.*	MS-03
FR-04-02	The bandwidth of the bus **should** allow the transfer of raw	MS-03

	EMG signals in real time. *Comment: EMG has a practical bandwidth of at most 1000 Hz; this figure may be significantly reduced in the context of prosthesis control.* *Real-time EMG transfer* may *be implied by future control schemes.*	
FR-04-03	The bus **shall** facilitate data transfer between any two connected nodes.	MS-01 MS-02-02 MS-02-03
FR-05	The bus protocol **shall** exploit known relationships and communication demands in order to minimise bus load and maximise user benefits.	MS-03-04
FR-05-01	Data elements with similar transfer rates from one and the same device **should** be sent in a single message to reduce overhead.	MS-03-04
FR-05-02	Bus messages **shall** be assigned priorities according to their relative time criticality. *Comment: Cyclic traffic related to EMG and motor control typically more critical than routine housekeeping, status information etc.*	MS-05-01-04
FR-05-03	Data elements with multiple receivers **should** be broadcast to all receivers simultaneously.	MS-03-04 MS-05-01-04-01
FR-06	The bus **shall** accommodate interoperability. *Comment: Interoperability means that one device may be swapped with another similar device with a minimum of reconfiguration requirements.*	MS-05-01-03 MS-05-02-02
FR-07	The bus **should** accommodate interchangeability. *Comment: Interchangeability means that one device may be swapped with another similar device* without any *reconfiguration requirements. This implies standardized mechanical, electrical and communication interfaces as well as standardised behaviour.* ***This is an extremely strict requirement, and will most likely*** **not** ***be realised within foreseeable future****. It is included here because it is of crucial importance for the future.*	MS-05-01-03 MS-05-02-02
FR-07-01	The bus **should** accommodate hot-swapping of identical or similar devices. *Comment: hot-swapping means a device is replaced while the system is powered-up and running, and that the new member automatically enters into the role of the device that is replaced.*	MS-05-01-03 MS-07-02
FR-08	The bus **shall** be able to communicate with a Service Controller (SC) when such is attached to the bus.	MS-05-02-01
FR-08-01	The SC **shall** be able so monitor the normal bus traffic.	MS-05-02-01
FR-08-02	The SC **shall** be able to read diagnostic information from the bus system. *Comment: This probably calls for a single "master" controller being responsible for collecting such info during operation and relaying it to the SC when solicited.* *Comment: Is there a need for assuming normal us operation* during *diagnostic read-out? Probably not.*	MS-05-02-03
FR-08-04	The SC **should** be able to download new firmware to any controller connected to the bus.	MS-05-02-04

	Comment: Subject to discussion; is this too ambitious? Each controller could alternatively have a dedicated programming port for this purpose, but download via the bus facilitates TRUE remote operation via an IP gateway etc.	
FR-08-05	A service person **shall** be able to set-up the prosthesis configuration via a SC connected to the bus.	MS-05-02-01
FR-08-05-01	The configuration set-up **should** be semi-automatic, with the bus system automatically collecting information about the devices and functionality present.	MS-05-02-01
FR-09	The system **shall** have a fail-safe behaviour.	MS-04-01 MS-05-01-01
FR-09-01	In case of an unrecoverable failure, the system **shall** assume a safe state. *Comment: Safety with respect to the user, e.g. the elbow should not suddenly jump to maximum flexion lest someone get hurt. Rather, it should be unpowered or held at a constant angle.*	MS-04-01 MS-05-01-01
FR-09-02	In the case of a temporary failure, all system components **shall** continually exhibit a safe behaviour. *Comment: Temporary failure in one part of the system might require temporary change in the behaviour of another component, e.g. during the performance of coordinated motion.*	MS-04-01 MS-05-01-01
FR-09-03	Each system components **shall** have a default "safe state" that is automatically assumed when communication fails or other critical errors occur. *Comment: The safe state may be different for different types of components, e.g. a hand and an elbow. Safe state should be specified for each device profile.*	Follows from FR-09-01 and FR-09-02
FR-10	At power-up, the system **shall** assume normal operation without any user interaction. *Comment: A precondition for this is that the system has previously been properly configured, cf.* FR-08-05. *All other initialisation etc. shall then be automatic when the system is powered-up.*	MS-04-02
FR-11	The bus **shall** exhibit electromagnetic compatibility with other prosthesis components and surroundings.	MS-07-01
FR-11-01	The bus **shall** be not induce excessive noise in measured EMG signals.	MS-07-01
FR-11-02	The bus **shall** be immune to the influence of domestic power lines, household equipment, commercial anti-theft systems (RFID etc.) and other "normal" sources of electrical noise.	MS-07-01