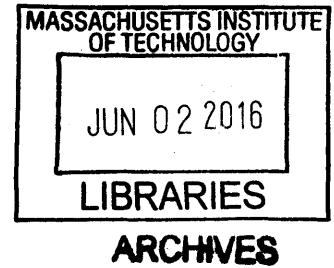


**Modular System Diagrams for Robotics and Their
Use in the MICA Project**

by

Craig Cheney

B.S., MIT (2014)



Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

© Massachusetts Institute of Technology 2016. All rights reserved.

Author **Signature redacted**
Department of Mechanical Engineering
May 6, 2016

Certified by **Signature redacted**
Ian W. Hunter
Hatsopoulos Professor of Mechanical Engineering
Thesis Supervisor

Accepted by **Signature redacted**
Rohan Abeyaratne
Chairman, Department Committee on Graduate Theses

Modular System Diagrams for Robotics and Their Use in the MICA Project

by

Craig Cheney

Submitted to the Department of Mechanical Engineering
on May 6, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

In this thesis, I propose a set of general rules for creating self-consistent System Diagrams across multiple domains. A System Diagram Prototype is then constructed and proposed for the specific use of diagramming robots. This robotic prototype is useful for teaching robotics and several examples of how robots can be diagrammed are given. The robotic prototype is then applied to creating sensors and generator modules in the MICA Project. The MICA (Measurement, Instrumentation, Control, and Analysis) Project is an ongoing research effort in MIT's BioInstrumentation Laboratory that aims to bring powerful, simple-to-use, wireless sensors and generators into the world of education. As education gets augmented by software-based teaching, it is essential that hardware continues and expands into educational demonstrations and lessons. A core tenet of the MICA project is that students learn best through hands-on, measurement-driven education. Real-life experiments can provide a deeper intuition and a stronger motivation, than purely abstract lessons. By having a set of easy-to-use sensors and generators, individual students can quickly create an experiment, measure the results, and then analyze the results. When coupling this individualized hardware approach with software teaching methods, a new method for teaching students of all ages and backgrounds is possible, one which embodies the MIT motto, *mens et manus*, mind and hand.

Thesis Supervisor: Ian W. Hunter

Title: Hatsopoulos Professor of Mechanical Engineering

Acknowledgments

First and foremost, I would like to thank Professor Ian Hunter for being an inspiration as an educator, an inventor, and an entrepreneur. He has been profoundly important to my time here at MIT, and I look forward to continuing my work with him in school and beyond.

The experience of being in the BioInstrumentation Laboratory is a one of a kind opportunity, and I am grateful to be surrounded by so many intelligent and passionate labmates; Ashin Modak, Ashley Raynal, Seyed Mirvakili, Nick Demas, John Liu, Anshul Signal, Geehoon park, Dr. Cathy Hogan, and Kate Melvin for all being an integral part of the research environment.

Of the many undergraduates who have helped contribute to my research, I would like commend two particular standouts, Eric Dahlseng, and Sam Edson.

Lastly, the person who has always supported me and helped me out with whatever I may need, my friend, colleague, co-educator, and girlfriend, the wonderful Danielle Class.

Contents

1	Rules of System Diagramming	21
1.1	System Diagramming Operators	21
1.1.1	Connectors	22
1.1.2	Ports	23
1.1.3	Tunnel Operators	25
1.1.4	Block Operators	25
1.1.5	Operator Summary	26
1.2	Prototypes of a System Diagram	26
1.2.1	Block Prototypes	28
1.2.2	Connector Prototypes	29
1.2.3	Port Prototypes	30
1.2.4	Tunnel Prototypes	30
1.2.5	Prototype Summary	30
1.3	Features of System Diagrams	33
1.3.1	Libraries	33
1.3.2	Block Groups	33
1.3.3	Connector Groups	37
2	System Diagramming for Robotics	39
2.1	Robot Prototype	40
2.2	Electrical prototype	42
2.2.1	Electrical System	42
2.2.2	Electrical Modules	44

2.2.3	Electrical Units	46
2.2.4	Electrical Subunits	47
2.2.5	Electrical Components	48
2.2.6	Finalized Electrical Prototypes	49
2.3	Mechanical Prototype	51
2.3.1	Bond Graphs	51
2.3.2	Mechanical System	55
2.3.3	Mechanical Modules	56
2.3.4	Mechanical Units	57
2.3.5	Mechanical Subunits	58
2.3.6	Mechanical Components	59
2.3.7	Finalized Mechanical Prototype	59
2.4	Software Prototype	61
3	Modeling a Robot from the System Diagramming Prototype	63
3.1	Electrical System for a “Charging Robot”	63
3.2	A Functional Robot Example	73
3.3	Electrical System	88
3.4	Software	96
3.5	Completed DriveBot Diagram	97
4	Using System Diagramming to Design MICA Blocks	99
4.1	System Diagramming	99
4.2	Electronics	101
4.2.1	Schematic Design	102
4.2.2	Board Design	103
4.2.3	Population	105
4.3	Mechanics	107
4.3.1	Form Factor and User Experience	107
4.4	Software	109
4.4.1	Embedded Software	109

4.4.2	MICA Mobile	110
4.4.3	Web Application	112
5	Conclusion	115

List of Figures

1-1	A block with a single input and a single output.	22
1-2	The output from Block A is laterally connected to in the input of Block B.	22
1-3	Block B is vertically connected to its parent block, Block A.	23
1-4	A block with three ports shown.	23
1-5	Block A has only an output port, and Block B has only an input port, meaning there is only one valid connection to be made.	24
1-6	The block on the left is valid as the connection types match the port types. The block on the right side is not valid as the connections do not match.	24
1-7	Block A exists in two domains, domain α and domain β . The domains are connected by a tunnel operator.	25
1-8	Block operators can change the type and number of connections passing through them.	26
1-9	The four operators of System Diagramming	26
1-10	The puzzle position on the left is analogous to a block prototype, and the two puzzle pieces on the right are analogs to blocks that can fill the prototype. Even though the two puzzle pieces can fill the same prototype, they are functionally different, as represented by differing colors.	27
1-11	Each of the standard operators has a corresponding prototype operator.	28

1-12	The block prototype in the top half of the figure represents a puzzle piece position. In the lower half, the three blocks represent the different puzzle pieces. Block Y may not fill in the prototype block because it does not have the necessary ports to connect to block B.	29
1-13	Block A is connected to blocks B and C by a connector prototype. . .	29
1-14	A port prototype is represented by a double line port. They can only be located along the edge of a parent block.	30
1-15	(Top) An example prototype diagram that uses all four types of prototype operators. (Bottom) A selection of blocks that may be used to fill the prototype.	31
1-16	The prototype of Figure 1-15 is filled in with blocks	32
1-17	The block diagram is functionally identical to Figure 1-16, regardless of the prototype.	32
1-18	A group of two blocks. The group is denoted by the dashed lines around the blocks.	34
1-19	Double dashed lines represent a block group prototype. On the right is a library of subunits that could be used to populate the subunit prototypes.	34
1-20	Groups can be populated from libraries either as individual blocks, or as an entire group. In this case subunits from the subunit library were used to populate the subunit prototypes	34
1-21	A collapsed version of a group with two inputs and two outputs. . . .	35
1-22	An example filter diagram with three prototype blocks. Blocks two and three are inside a group prototype. Blocks can be filled from the library on the right.	35
1-23	A filled in version of Figure 1-22	36
1-24	The functional equivalent of Figure 1-23, drawn without the group. .	36
1-25	An incorrect use of a block prototype, where a group prototype should have been used instead. None of the blocks in the library fit into Filter Components one or two, as the library only contains subunits.	37

1-26	Two blocks with four connections between them. To save space and time the connectors could be drawn in a group, as in Figure 1-27 . . .	37
1-27	Two blocks with a group of connectors between them. Equivalent to Figure 1-26	38
1-28	Two block prototypes connected by a group of 4 prototype connectors.	38
2-1	A collapsed block prototype that any robot block can populate. . . .	40
2-2	Based on the definition of the robot, the Robot Prototype must contain three system level blocks, one for each domain - mechanical, electrical, and software	41
2-3	There are six different vertically connected levels to the the Robotic Prototype.	41
2-4	There are two connection types (signal and power), and three directions (in, out, bidirectional), meaning that there are six ports into the Electrical System Prototype.	43
2-5	Each of the six system ports has a corresponding module. Any connection entering the Electrical System must be routed to the appropriate module based on the connection's type and direction.	44
2-6	Each module contains six units, one for each type and direction of connection. Units within the same module may be freely connected to each other, regardless of connection type and direction.	46
2-7	As each unit contains an arbitrary number of subunits, there are no prototypical restrictions on which subunits can connect to the unit ports.	47
2-8	Subunits are composed of one or more components. Components are the lowest level block in the Robot Prototype and represent discrete electrical items such as integrated circuits, resistors, capacitors, and inductors.	48
2-9	A unit prototype diagram is drawn with expanded subunits.	49
2-10	The result of substituting expanded units into Figure 2-6.	50
2-11	The fully expanded system diagram for the electrical system prototype.	50

2-12	The System Diagram, electrical schematic, and Bond Graph for a resistor connected to a battery.	52
2-13	Comparison of the different connection types in the Mechanical and Electrical domain	53
2-14	The System Diagram, electrical schematic, and Bond Graph for a current source driving a DC motor attached to a frictionless wheel. . . .	54
2-15	The System Diagram, electrical schematic, and Bond Graph for a battery driving a non-ideal DC motor attached to a wheel with friction. .	55
2-16	The System prototype for mechanical systems. Only bidirectional connections are allowed in the mechanical domain, meaning that there are fewer ports and modules than in the Electrical domain.	56
2-17	In the Mechanical Module Prototype, only the Dynamic Unit can connect to the Dynamic Port, and only the Static Unit can connect to the static port.	57
2-18	The Mechanical Unit Prototype is composed of an arbitrary number of subunits. There are no restrictions on which subunits can connect the Unit's ports.	58
2-19	The lowest level in the Mechanical System Prototype is the component level. An arbitrary number of components make up the Mechanical Subunit Prototype.	59
2-20	Expanding each subunit results in the expanded unit diagram.	60
2-21	Expanding each unit results in the expanded module diagram.	60
2-22	Expanding each module results in the expanded system diagram.	61
3-1	Empty System Diagram Prototype for ChargeBot.	64
3-2	Collapsed Electrical System block for ChargeBot.	64
3-3	Expanded Electrical System of ChargeBot. The battery charger inputs may only be routed to the Energy Module based on their type and directionality.	65

3-4	Expanded Energy Module block for ChargeBot. The inputs routing to the Energy Unit is mandated by the Robot Prototype.	66
3-5	Energy Unit of ChargeBot. Note how there are no longer the double lines around the subunits, as there is no specific prototype that the two subunits are filling.	66
3-6	The LTC4065 Battery Charger IC is the critical block of the Battery Charging Subunit.	67
3-7	The battery subunit of ChargeBot. Normally the battery would have an output to power the rest of the system.	68
3-8	Updated version of the Energy Unit (v1.1), now reflecting the LED output signal.	68
3-9	LED output must pass through the Control Unit before it can exit through the Signal-Out port.	69
3-10	A Pass-Through block. The LED signal passes through the Control Unit so that it can be routed out of the Signal-Out Module port. . . .	70
3-11	Incorrect routing of the LED Output signal. Signals may only exit through the system Signal-Out port from the Control Module.	70
3-12	Correct routing of the LED Output signal. In this case the Control Module is a pass-through block.	71
3-13	LED Out passing though the Control Module in order to exit the system.	71
3-14	Pass-Through Sensing Unit to route out the LED connection.	72
3-15	Pass-Through Control Unit to route out the LED connection.	72
3-16	Complete Electrical System diagram. Note that this is version V1.1, instead of V1.0 like Figure 3-2	73
3-17	A CAD model rendering of the DriveBot.	74
3-18	Starting point for the DriveBot System Diagram - a blank robotic prototype	74
3-19	A system diagram for DriveBot drawn outside the context of the Robotic Prototype.	74
3-20	A system diagram for DriveBot in the context of the Robotic Prototype.	75

3-21	A close up of the Slider on the front of the robot.	76
3-22	System Diagram for the Slider to Chassis connection outside the context of a prototype.	76
3-23	The Static and Dynamic Modules of DriveBot, in the context of the robotic framework.	77
3-24	System diagram of the Chassis Unit of the Chassis Module for DriveBot.	78
3-25	Expanded subunit diagrams for the chassis of DriveBot.	79
3-26	System Diagram for the Dynamic Motor Unit, which occupies the Dynamic Unit of DriveBot mechanical system.	80
3-27	A basic motor and wheel diagram for DriveBot.	81
3-28	A comparison of a DC motor system in the Electrical and Mechanical domains, outside of a prototype.	82
3-29	Chassis, with battery, motors, and electrical connectors, outside of a prototype context.	82
3-30	The System level mechanical diagram with the DC motor Module expanded.	83
3-31	The two blocks contained within the DC motor Module. Note the four Pass-Through connections that go through the Static Unit and into the Chassis Module.	84
3-32	DC motor subunit diagram with the motor leads connected to alligator clips.	84
3-33	The electrical domain system diagram of DriveBot, outside the prototype context.	85
3-34	Electrical system diagram for DriveBot, this time in the context of the robot prototype.	85
3-35	The Battery and DC motor electrical modules, inside the context of the robot prototype.	86
3-36	The critical electrical units for DriveBot v1.1.	87

3-37	Finished System diagram for version 1.1 of DriveBot. Notice that the Software system prototype is still empty, meaning that DriveBot is not yet a Robot. DriveBot will meet the definition for a robot in a version presented later in this chapter.	87
3-38	A microcontroller is added in the Control Unit of the Control Module.	88
3-39	The critical block of the Control Module is the Control Unit, which contains a microcontroller. Only Pass-Through units are required elsewhere in the Module.	89
3-40	A dual H-bridge power amplifier is used to drive the two motors bidirectionally.	90
3-41	Two pass-through units are needed to support the critical Actuation Unit.	90
3-42	The Linear Regulator subunit produces a regulated output from an unregulated input. A power indication LED is used to show when the robot is powered on.	91
3-43	The Power Module for DriveBot.	91
3-44	The battery, circuit protection, and power switch subunits.	92
3-45	Completed energy module	93
3-46	The communication module enables a user to send commands via a mobile device. The commands are then sent to the control module for execution.	93
3-47	Only one pass-through block, supplying power, is needed for the Communication Module.	94
3-48	The Battery Measure block converts a power type to a signal type.	95
3-49	Two Pass-Through blocks are required in the Sensing Module.	95
3-50	Completed Electrical system diagram from DriveBot v1.1	96
3-51	Diagram of the software system. The prototype for this block is empty by default.	97

3-52	Final system diagram for DriveBot. Version 1.2 of DriveBot meets the definition of a robot, as it has mechanical, electrical, and software systems, as well as a controlled output.	98
4-1	A MICA Inertial Measurement Unit (IMU) block. The two gold plated magnets can be seen at the top of the block, and the battery can be seen through the translucent enclosure.	100
4-2	MICA blocks can be snapped together via electrically conductive magnets, to expand the functionality of the MICA system.	101
4-3	A MICA IMU with the top cover removed.	102
4-4	The top level circuit schematic in Altium, which is the same as the top level electrical System Diagram for the MICA IMU.	104
4-5	A isometric view of the final CAD model of the MICA IMU circuit board.	105
4-6	The top (left) and bottom (right) of the MICA IMU circuit board. Subunits are denoted by white boxes around them on the silkscreen layer.	105
4-7	An orange Kapton stencil is adhered to the unpopulated MICA IMU circuit board, seen through a microscope. The stencil is laser cut based on the mechanical component models, and their placement on the PCB.	106
4-8	A CAD model of the final MICA IMU. The enclosure is 3D printed using stereolithography.	107
4-9	(Left) The final MICA IMU and (right) a previous version. Both blocks have gold plated permanent magnets for mechanical mounting and battery charging.	108
4-10	A screenshot of the schematic in PSoC Creator that controls the MICA IMU. The schematic shown closely relates to the software system diagram of the MICA IMU.	110
4-11	The software System Diagram for the sensor data when the MICA block is connected to MICA Mobile.	111

4-12	Acceleration data are plotted in MICA Mobile as a previous version of a MICA IMU is shaken.	111
4-13	The software System Diagram for the sensor data when the MICA block is connected to a BLE-USB dongle and MICA Chrome.	112
4-14	As the MICA IMU is moved acceleration data is plotted in real time on the MICA web application.	113

Chapter 1

Rules of System Diagramming

Block diagrams are a familiar tool used in engineering domains. When used in controls theory, block diagramming has a rigid set of rules for how blocks interact with one another. However, when used to design a system, a similar set of rules does not exist. In this chapter, I propose a set of rules that allow diagrams to be consistent across multiple domains thus enabling systems to be fully defined by their System Diagrams.

1.1 System Diagramming Operators

The fundamental unit of a System Diagram is a block, which represents an abstract functioning piece of a system. Block inputs are represented by an arrow pointing into the block, while outputs of the block are represented by arrows pointing away from the blocks. Bidirectional connections can also exist, denoted by a line with an arrow at both ends. By convention, inputs generally point into the left side of block and outputs point out of the right side of the block. This convention is observed for diagram readability, however it is the arrows that formally define the directionality of a connection.

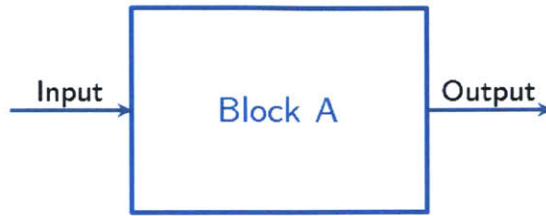


Figure 1-1: A block with a single input and a single output.

1.1.1 Connectors

Non-trivial system diagrams will contain more than one block. In order to get a two block system to function, there must be a way of wiring the inputs and outputs of one block to another. To do this, we can simply connect the output of Block A to the input of Block B. A connection between two blocks, by this method, is called a *Lateral Connection*. An example of a lateral connection is shown in Figure 1-2.

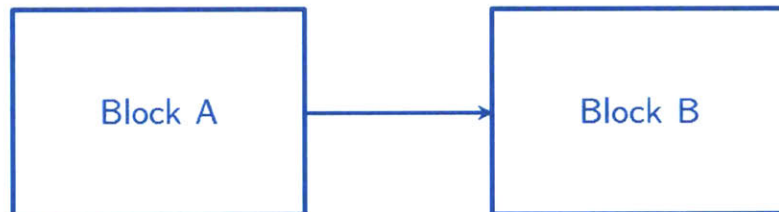


Figure 1-2: The output from Block A is laterally connected to in the input of Block B.

Lateral connections are useful when one wants to concatenate the functionality of blocks. They are not useful in describing how a given block functions. For all blocks, it is possible to analyze the block and learn how the block is taking its inputs and converting them to outputs. This is analogous to taking the lid off a box and seeing what is inside. These connections are called *Vertical Connections*. Vertical connections are represented by a child block nested within a parent block. When a vertical connection takes place, the child block of the parent is on a lower *Level* than its parent. Blocks on different levels may not laterally connect to each other; rather they must go through a *port*.

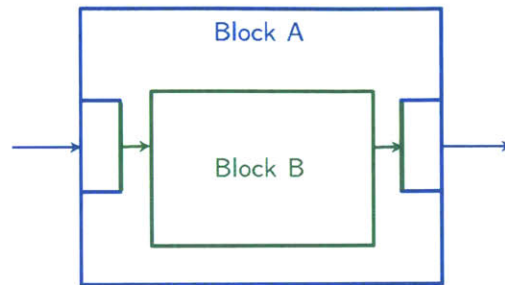


Figure 1-3: Block B is vertically connected to its parent block, Block A.

Note that Block B in Figure 1-3 is not shown with ports. The existence of the ports is implied, and in fact mandated. However, users may opt to keep blocks *closed* and not represent any of the internal components. Given blocks A and B, how does one know that block A has the ability to be connected to block B? This question motivates the use of *Ports*.

1.1.2 Ports

Ports are the mechanisms by which connections enter and exit all blocks. Any time that a connection crosses through the boundary of a block, it must do so through a port. A port is represented by a nested block that shares one edge with its parent. Ports must specify the directionality of the connection they accept as an inputs, outputs, or bidirectional. By convention, input ports are on the left of blocks, output ports are on the right blocks, and bidirectional ports are on the top and/or bottom of blocks, as seen in Figure 1-4.

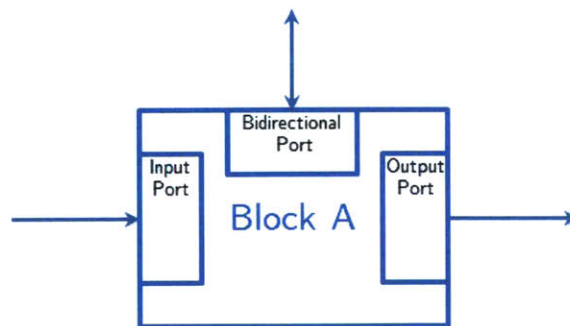


Figure 1-4: A block with three ports shown.

The directionality of ports imposes a set of constraints on the wires coming from them. Revisiting the example depicted in Figure 1-2, if one was given blocks A and B that had ports drawn on them, as in Figure 1-5, it is immediately apparent that there is only one valid configuration for wiring the blocks.

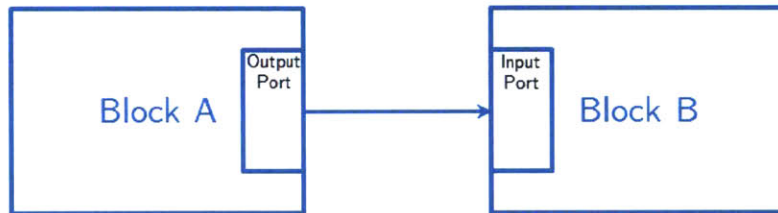


Figure 1-5: Block A has only an output port, and Block B has only an input port, meaning there is only one valid connection to be made.

However, in general, the directionality of a connection is not enough to tell whether or not a connection can enter or exit a given port. In addition to the directionality, the *Type* of the connection must match the *Type* of the port. Types can be arbitrary, but are specified within a given domain. For System Diagrams representing electrical circuits, for example, we will see that it is convenient to have two types, *Signal* and *Power*. While not necessary, the use of dashed lines to represent types can dramatically improve the readability of System Diagrams. Figure 1-6 shows a valid connection to a port, and an invalid connection.

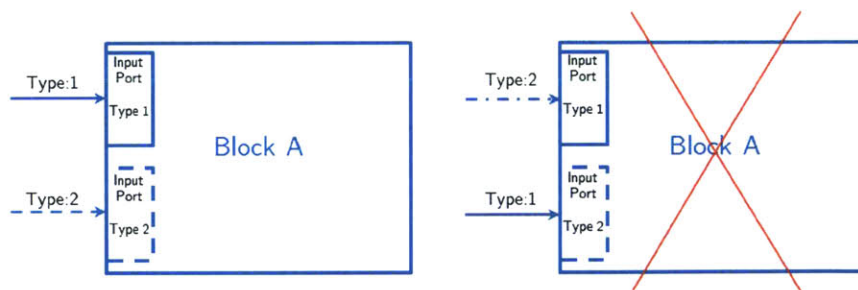


Figure 1-6: The block on the left is valid as the connection types match the port types. The block on the right side is not valid as the connections do not match.

1.1.3 Tunnel Operators

When blocks need to be connected between different Domains, a *Tunnel* connection is used. Tunnel connections exist between two blocks that represent the same physical object in two different domains. For example, an electromechanical device, such as a power switch, will have a tunnel connecting its block in the electrical domain with its block in the mechanical domain, shown in Figure 1-7. A Tunnel is represented by an arrow pointing into, or out of, an ellipse, depending on the direction of the Tunnel. The domain of the block that uses the Tunnel is written on the side of the arrow opposite the ellipse. The domain that is being Tunneled into is written on the same side as the ellipse.

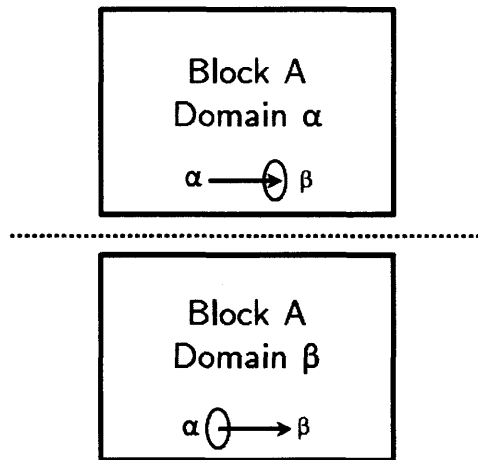


Figure 1-7: Block A exists in two domains, domain α and domain β . The domains are connected by a tunnel operator.

1.1.4 Block Operators

The last operator that exists in the system diagram is the block itself. A block can convert the *type* of an input to a different type output. However, an input does not necessarily change types just because it goes through a block, it depends on the specific implementation of the block in question. Additionally, the inputs and outputs of a block are not required to map in a one to one manner with the output. One input of type A can map to an output of type A and an output of type B . Examples

of this can be seen in Figure 1-8.

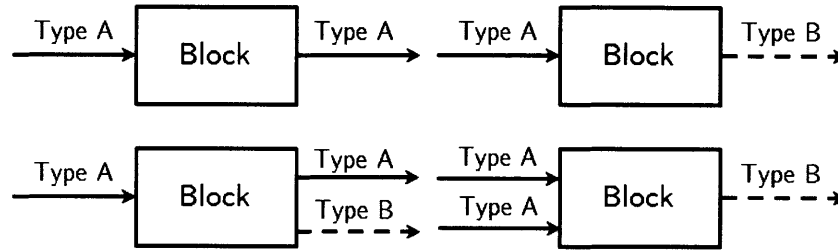


Figure 1-8: Block operators can change the type and number of connections passing through them.

1.1.5 Operator Summary

A summary of the block diagram operators may be seen in Figure 1-9.

Operator	Variable	Connection Name	Symbol
Block	Type	Block	
Connection	Block	Lateral	
Port	Level	Vertical	
Tunnel	Domain	Tunnel	

Figure 1-9: The four operators of System Diagramming

1.2 Prototypes of a System Diagram

A major advantage of System Diagramming, beyond representing a system, is that it can be used to speed up the design process through standardization. A System

Diagram whose form gets used over and over again is well suited to be turned into a *System Diagram Prototype*.

System Diagram Prototypes are placeholders for system diagrams that enforce specific design restrictions. For an analogy to a System Diagram Prototype, imagine a puzzle. The pieces of the puzzle are the blocks for the system, and the place that the puzzle pieces fit into are the prototypes. Different puzzle pieces are characterized by the color, or image, on them (the specific block they are representing) and their shape (which prototype the piece fits into). In Figure 1-10, there are two puzzle pieces, a red one and a blue one. Both of them fit into the puzzle piece place on the left, but a different system is produced depending on which color piece is inserted.

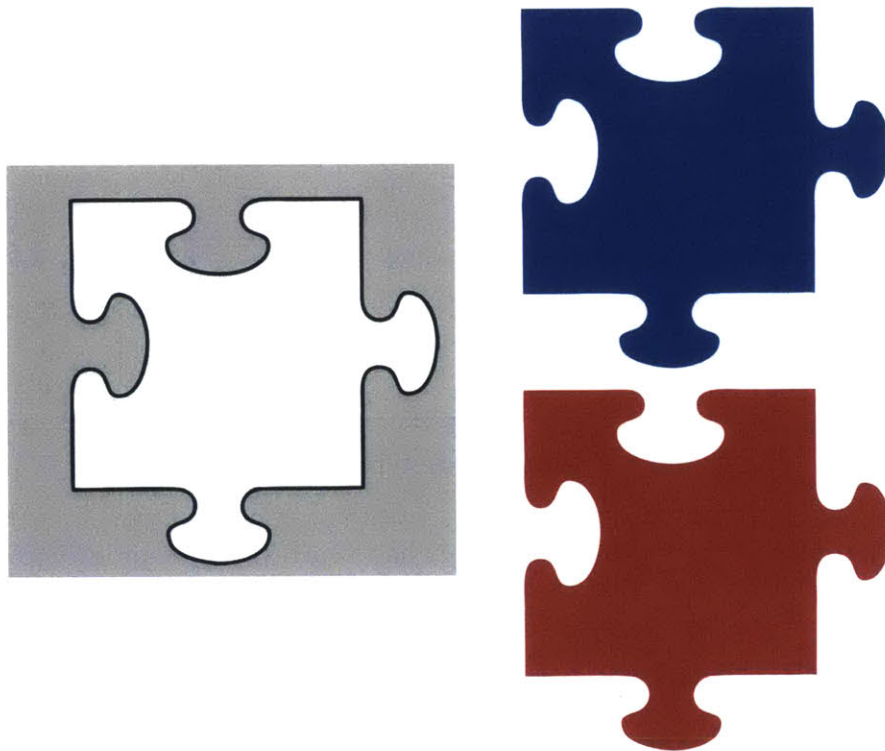


Figure 1-10: The puzzle position on the left is analogous to a block prototype, and the two puzzle pieces on the right are analogs to blocks that can fill the prototype. Even though the two puzzle pieces can fill the same prototype, they are functionally different, as represented by differing colors.

In system diagramming, prototypes define the inputs, outputs, and level that

characterize a block, as well as establish the overall role the block plays within in the larger system. Many blocks may be compatible with a given block prototype, and a given block may be compatible with many block prototypes. It is up to the system designer to create the desired functionality of a system based on the blocks they choose to fill in the block prototypes.

The four operators that were described in the previous section have direct prototype analogs. Prototype operators are represented by double lines, where a standard operator would have a single line. Prototype operators have the ability to be filled with standard operators. The four prototype operators can be seen in Figure 1-11.

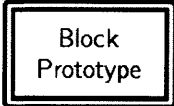



Operator	Prototype	Symbol
Block	Block Prototype	
Wire	Wire Prototype	
Port	Port Prototype	
Tunnel	Tunnel Prototype	

Figure 1-11: Each of the standard operators has a corresponding prototype operator.

1.2.1 Block Prototypes

The most common prototype operator is the *block prototype*. In the top half of Figure 1-12, there are two standard blocks, A and B, connected to a prototype block. Recalling the puzzle piece analogy presented earlier, the prototype block represents a puzzle position. In the bottom half of the example, there are three different blocks, X, Y and Z, that are available to fill in the prototype block, which are analogous to the puzzle pieces. Only blocks X and Z are suitable to fill in the prototype block, because block Y has no output port, which is necessary to create the connection that

goes to Block B.

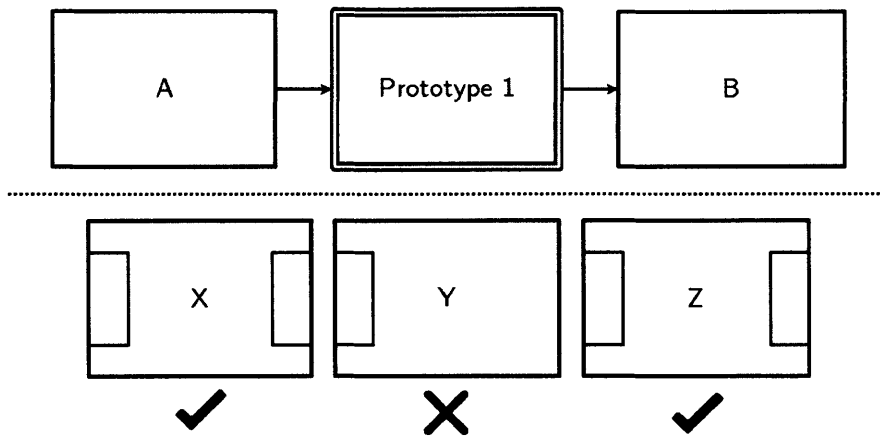


Figure 1-12: The block prototype in the top half of the figure represents a puzzle piece position. In the lower half, the three blocks represent the different puzzle pieces. Block Y may not fill in the prototype block because it does not have the necessary ports to connect to block B.

1.2.2 Connector Prototypes

When a block has the ability to connect to other blocks based on the designer's preference, a *Connector Prototype* should be used. A connector prototype is indicated by a double line with an arrow denoting directionality of the connection. As shown in Figure 1-13, depending on which one of the two connector prototypes is filled in with a connector, block A could be laterally connected to both, either, or neither of blocks B and C. This is dependent on how the designer chooses to implement the system.

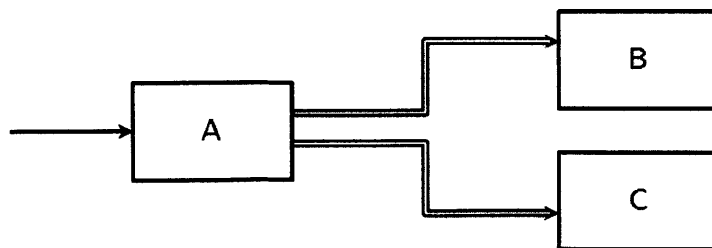


Figure 1-13: Block A is connected to blocks B and C by a connector prototype.

1.2.3 Port Prototypes

A port prototype can be used in a System Diagram when the designer wants to incorporate a degree of flexibility around which inputs and outputs will branch off a block. A port prototype is indicated by a double line port, seen in Figure 1-14.

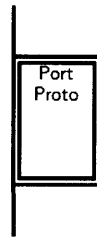


Figure 1-14: A port prototype is represented by a double line port. They can only be located along the edge of a parent block.

1.2.4 Tunnel Prototypes

A block or block prototype may contain a Tunnel Prototype. If a block prototype contains a tunnel prototype, it indicates that any block used to populate the block prototype must contain a tunnel, and therefore connect to a different domain. If a standard block contains the tunnel prototype, it indicates that the block has the option of connecting to a different domain.

1.2.5 Prototype Summary

Prototypes are a crucial part of System Diagramming that allow designers to indicate desired functionality without having to specify how systems achieve their functionality. Figure 1-15 is an example of a prototype diagram that uses all four operators. Below the diagram is a selection of blocks that can be used to populate the various block prototypes. There is a tremendous amount of flexibility in populating the various operators of the diagram, and different functionality can be achieved based on the specific case. Only block D can satisfy the tunnel prototype requirement of block prototype 4. Blocks A and C may be placed in positions 1, 2, or 3, although

placement in positions 1 or 3 would restrict which ports could be used. Block B may be placed in either position 1 or 3.

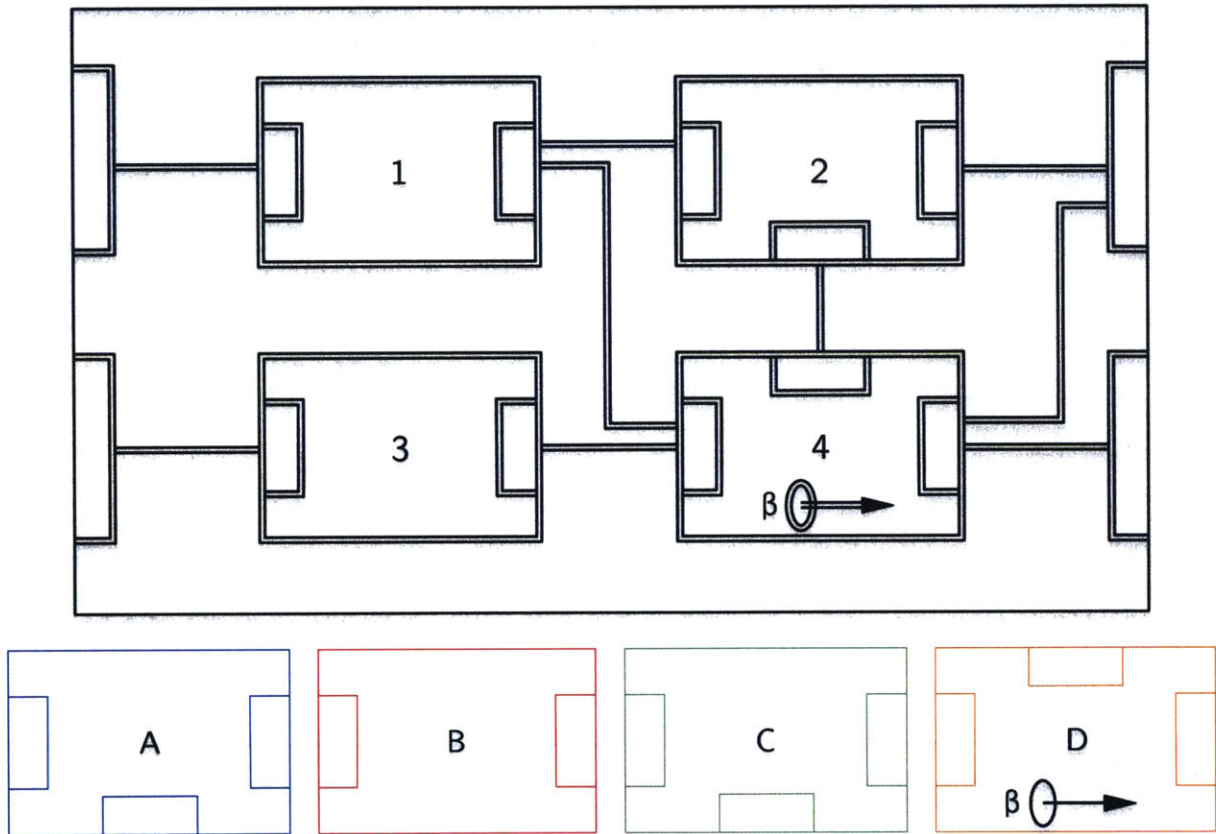


Figure 1-15: (Top) An example prototype diagram that uses all four types of prototype operators. (Bottom) A selection of blocks that may be used to fill the prototype.

In Figure 1-16, the blocks were placed into the prototype. Note that not all block prototypes were filled, and not all the connector or port prototypes were filled, both of which are valid. The presence of a prototype does not change the functionality of the resulting system diagram. Figure 1-17 is functionally identical to Figure 1-16, even though all the prototype elements were removed.

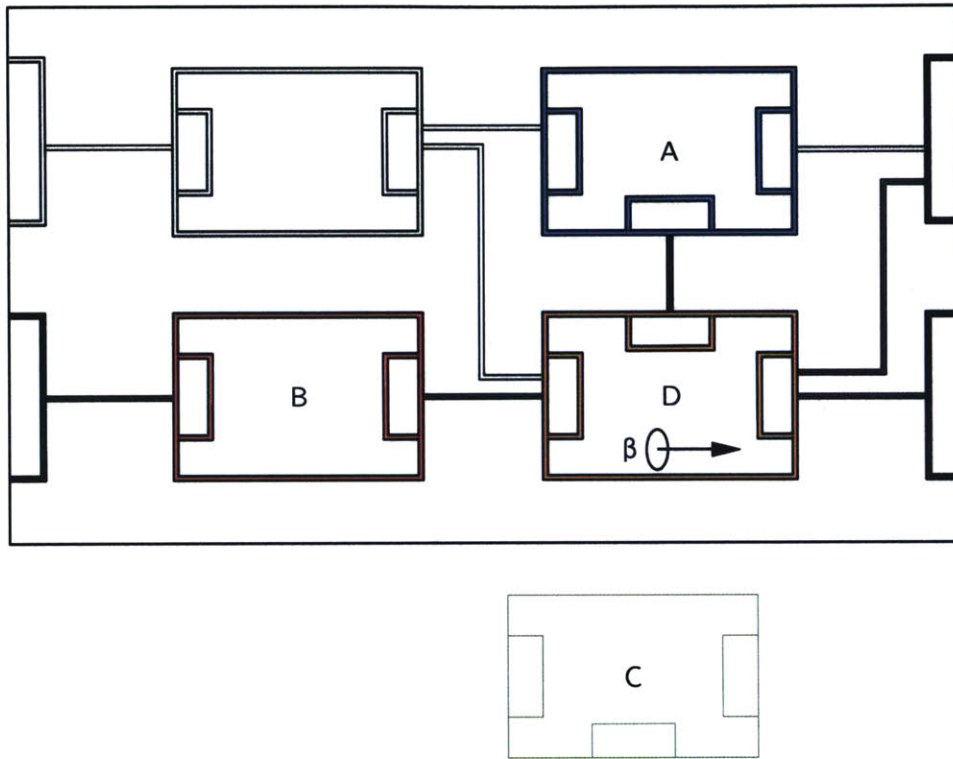


Figure 1-16: The prototype of Figure 1-15 is filled in with blocks

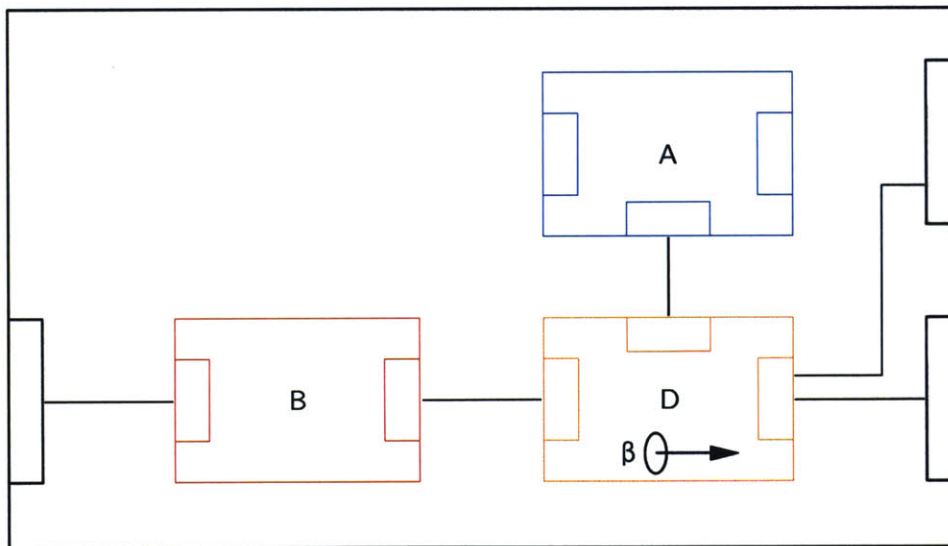


Figure 1-17: The block diagram is functionally identical to Figure 1-16, regardless of the prototype.

1.3 Features of System Diagrams

1.3.1 Libraries

An invaluable aspect of designing with System Diagramming is building up a library of blocks that can then be inserted into prototypes, and reused again and again. A necessary requirement for compiling a library, is that all blocks have an associated version number. As with any design, the ability to change, revise, and alter is crucial, the mechanisms that facilitate this must be built into the system. Blocks that are being inserted into prototypes need version numbers, but so do the prototype blocks themselves. This is true at all levels, including the system level. Additionally, the System Diagramming framework itself needs a version number to account for the inevitable change of the framework rules. When a block has been placed into a library, the system designers no longer have to concern themselves with how the expanded block functions. They only need to look at the inputs, outputs, and a descriptions of the block to know if it suitable for their purposes.

1.3.2 Block Groups

Block Groups can be used to organize multiple blocks on the same level within a system diagram. This is distinct from creating a block that contains the target blocks, as groups do not contain any ports. Therefore, any connections that go into groups do not change levels. A block group is denoted in a system diagram by a dashed line around a group of blocks, as seen in Figure 1-18. Groups can be either blocks, or prototypes, indicated by a double dashed line, as seen in Figure 1-19. Groups can be filled in from libraries, either by filling in the entire group, or by filling individual blocks of the group, as in Figure 1-20. Groups can be collapsed (Figure 1-21) and expanded just like blocks. The main purpose of block groups is to save space on diagrams, and for logical organization. All blocks inside a group must be on the same level.

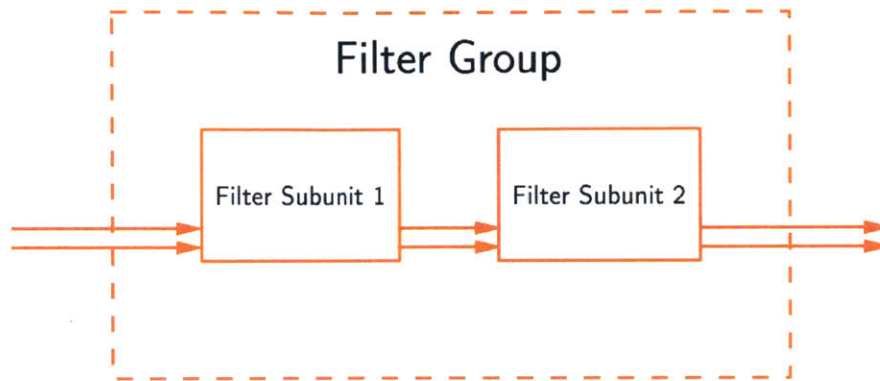


Figure 1-18: A group of two blocks. The group is denoted by the dashed lines around the blocks.

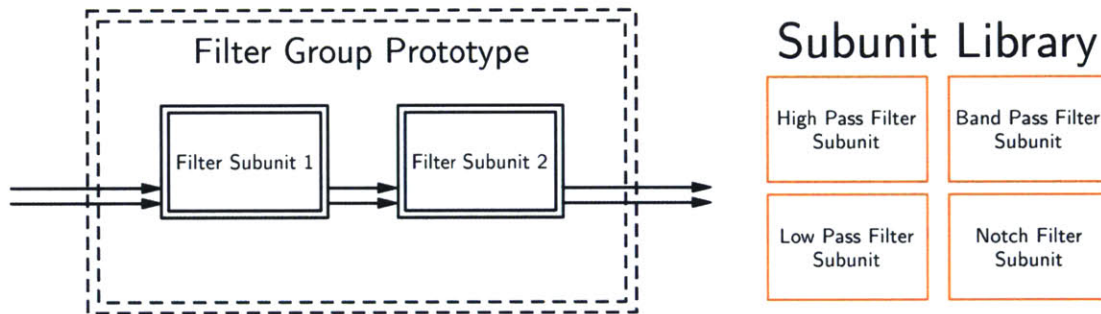


Figure 1-19: Double dashed lines represent a block group prototype. On the right is a library of subunits that could be used to populate the subunit prototypes.

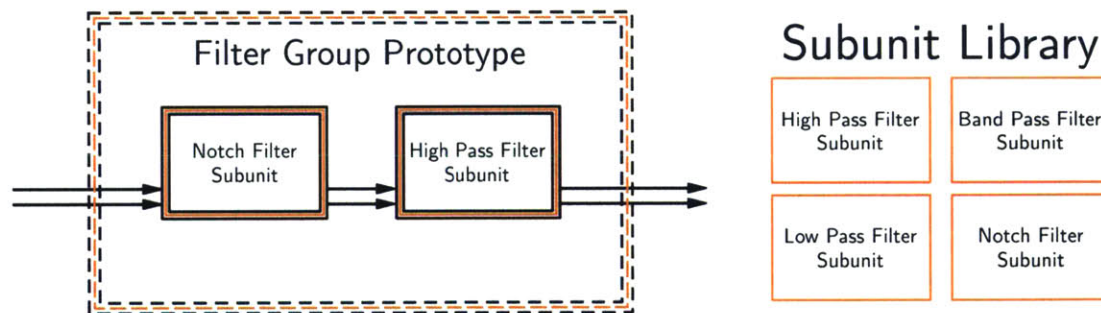


Figure 1-20: Groups can be populated from libraries either as individual blocks, or as an entire group. In this case subunits from the subunit library were used to populate the subunit prototypes



Figure 1-21: A collapsed version of a group with two inputs and two outputs.

A group is distinct from a block in that it allows for collapsing various parts of the diagram without changing the level of the connections. Consider the following example, depicted in Figure 1-22. There are three filter subunit prototypes, with prototypes two and three inside the group and prototype one outside. When the prototypes are populated from the library, a resulting diagram could be Figure 1-23. Which is the functional equivalent of Figure 1-24.

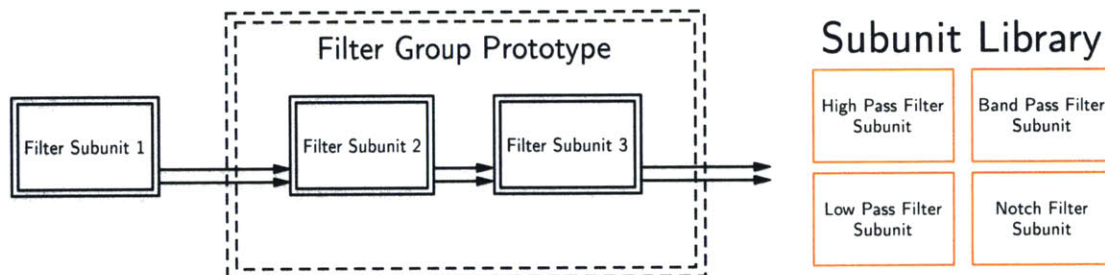


Figure 1-22: An example filter diagram with three prototype blocks. Blocks two and three are inside a group prototype. Blocks can be filled from the library on the right.

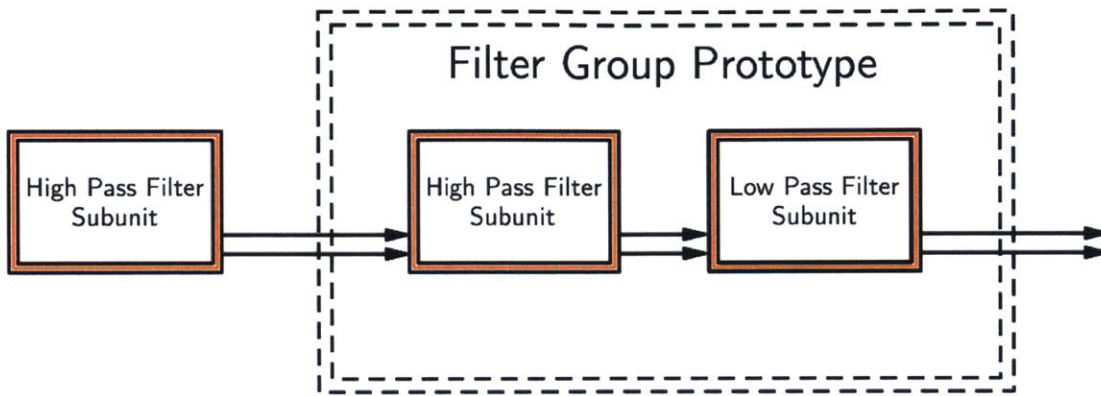


Figure 1-23: A filled in version of Figure 1-22

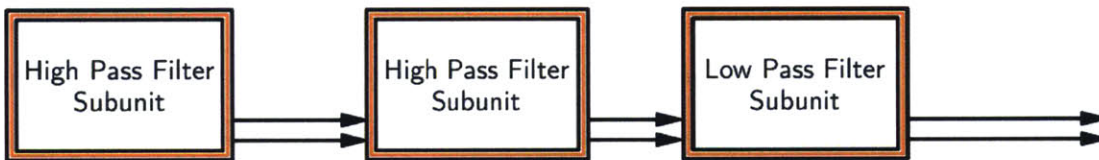


Figure 1-24: The functional equivalent of Figure 1-23, drawn without the group.

Now consider if a *block* prototype had been used instead of a *group* prototype. The result, shown in Figure 1-25, is that what were intended to be Filter Subunits two and three are now vertically connected to the Filter “Group” Subunit, meaning that they are not on the same level as Filter Subunit 1. Connections from Filter Subunit 1 pass through a port and must drop down a level. The result is that the high pass filter and low pass filter subunit no longer fit into their intended spots, and it is impossible to recreate the intended diagram of Figure 1-24.

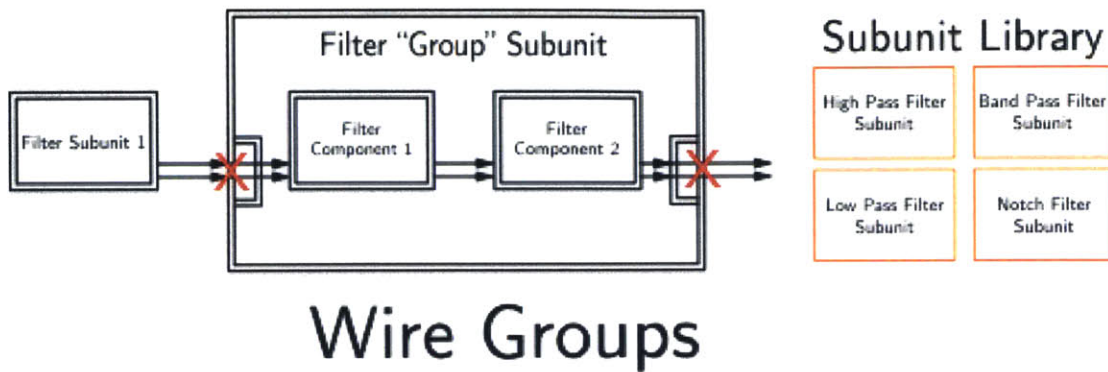


Figure 1-25: An incorrect use of a block prototype, where a group prototype should have been used instead. None of the blocks in the library fit into Filter Components one or two, as the library only contains subunits.

1.3.3 Connector Groups

Just like block groups, connector groups can be used for simplicity and space saving reasons. Connector groups are denoted by a single connector with a slash and then a number representing the number of connectors in the group. All connectors in a group must be of the same type. Figure 1-26 could be drawn as a group (Figure 1-27), which can be quite useful when the number of connectors gets large. Prototype connector groups can be used as well, as in Figure 1-28.



Figure 1-26: Two blocks with four connections between them. To save space and time the connectors could be drawn in a group, as in Figure 1-27



Figure 1-27: Two blocks with a group of connectors between them. Equivalent to Figure 1-26



Figure 1-28: Two block prototypes connected by a group of 4 prototype connectors.

Chapter 2

System Diagramming for Robotics

While System Diagramming is a generic tool and not specific to any one subject, it is highly useful for designing in the field of robotics. Before proceeding, it is necessary to define “robotics”. The definition that will be used in this paper is as follows: A robot is a device that uses a combination of electrical, mechanical, and software devices to produce a controlled output. This definition is sufficiently broad to include machines traditionally thought of as robots, such as humanoid devices like Boston Dynamics’ Atlas robot [2], but also includes devices less typically considered robots, like a turn-table microwave oven.

Presented in this chapter is a System Diagramming Prototype that can be used to represent devices that meet the definition for a robot. The prototype is meant to be general enough so as to not limit the designer’s freedom, but simultaneously provide guidelines and standardization that can enable good practices. A goal of using System Diagramming for robotics is to have a diagram that fully represents the system across the three domains in question: electronics, mechanics, and software. Furthermore, once the system diagram has been created, it can be used as the input to various CAD packages. For example, the electrical system diagram is exactly the same as the electrical schematic that is used in an electronic CAD package. It is often easier for designers to begin designing a device not by laying out the blocks necessary for the machine to function, but rather by laying out the *block prototypes*. A block represents a specific *implementation* of desired function, whereas a block

prototype represents the desired functionality. By standardizing on a prototype, the first step of designing a robot can be largely eliminated, allowing the designer to begin populating the prototype design with blocks from a library. As new robots are designed, the number of blocks in the library will grow allowing designers to reuse more and more of their past designs. This growing library will speed up the overall design process.

2.1 Robot Prototype

The robotic prototype will start with a single block prototype that represents the final robot. The robot has prototype connections for the inputs and outputs for the machine. Starting off, the initial collapsed prototype shown in Figure 2-1 is incomplete, so it will be labeled as version 0.1.

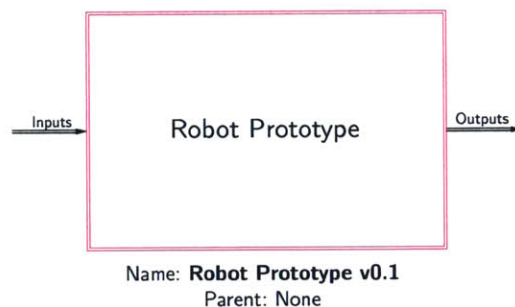


Figure 2-1: A collapsed block prototype that any robot block can populate.

Using the definition of a robot, expanding the robot prototype in Figure 2-1 should show a block prototype in each of the three domains - mechanical, electrical, and software. The level beneath the “Robot Level” is the “System Level”. Therefore the three block prototypes contained within the Robot Prototype (Figure 2-2) are the Mechanical System Prototype, the Electrical System Prototype, and the Software System Prototype, each belonging to their respective domain. These are added to the expanded version of Figure 2-2.

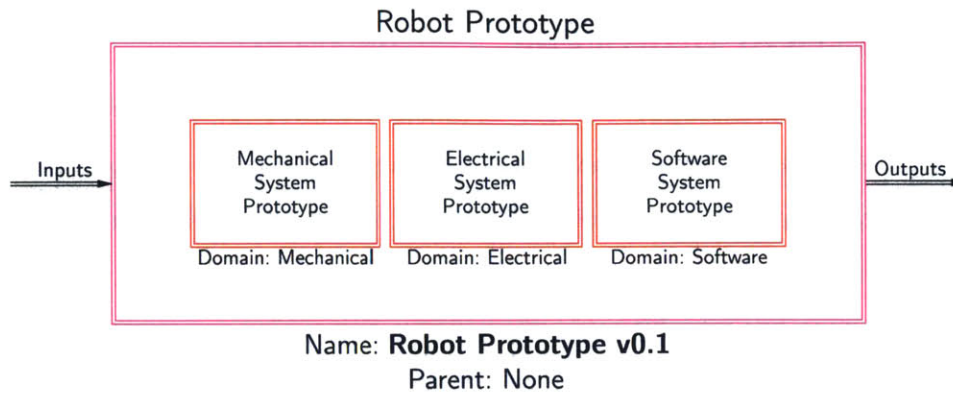


Figure 2-2: Based on the definition of the robot, the Robot Prototype must contain three system level blocks, one for each domain - mechanical, electrical, and software

In the next sections, the three domains will be explored from the System Level, down to the bottom-most level covered in the Robot Prototype: the Component Level. In total there are six different vertically connected levels of the Robot prototype, each one designated a color for clarity. From the top to bottom, the levels are: Robot (magenta), System (red), Module (blue), Unit (green), Subunit (orange), and Component (black). The different levels of the robotic prototype are shown in Figure 2-3.

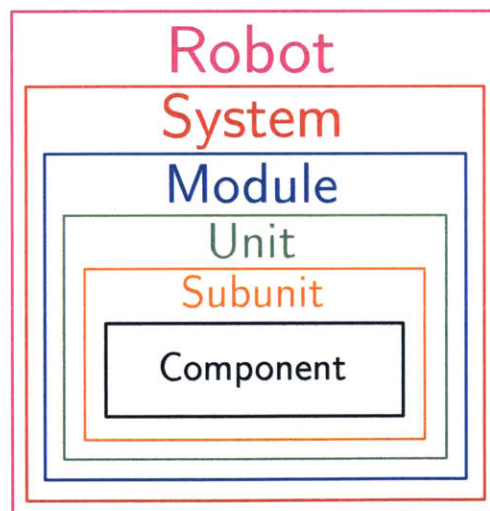


Figure 2-3: There are six different vertically connected levels to the the Robotic Prototype.

2.2 Electrical prototype

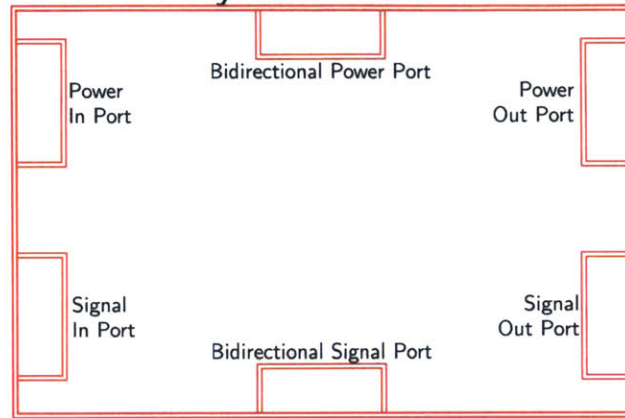
In the proposed Robotic Prototype, the electrical system has by far the highest level of granularity, and will be the system discussed first. System diagramming can be thought of as the act of telling the story of the inputs and outputs. If one follows the inputs through the various levels of a system, the inputs eventually come out on the other side of the system as outputs. If all of the inputs and outputs are followed through the system, then the system is fully defined. This means that to start laying out any block or prototype block, the different *types* of inputs and outputs must be defined.

A system diagram connection in the electrical domain represents an electrical connection from one block to another. The state of the connector is the electrical potential, and as a result, current flows through the connections. Two *types* of connectors exist in the prototype, based on the relative magnitude of the current flowing through them. For a relatively small current flowing, the connection is of type *Signal*, and for relatively large current the connection is of type *Power*. The magnitude is relative to the block that is consuming current, but can be thought of as Signal Connections delivering information, whereas Power Connections deliver energy.

2.2.1 Electrical System

In addition to the two types of connections in the electrical domain, there are three different directions that a given connection can assume; an input, an output, or a bidirectional line. As a result, there are six different combinations of types and directions for electrical connectors. Connections can only enter a block through a port of the matching type and direction meaning that the Electrical System must have six ports. Expanding the Electrical System Prototype shown in Figure 2-2 reveals the six ports available, shown in Figure 2-4.

Robot Prototype - Electrical Domain System Level



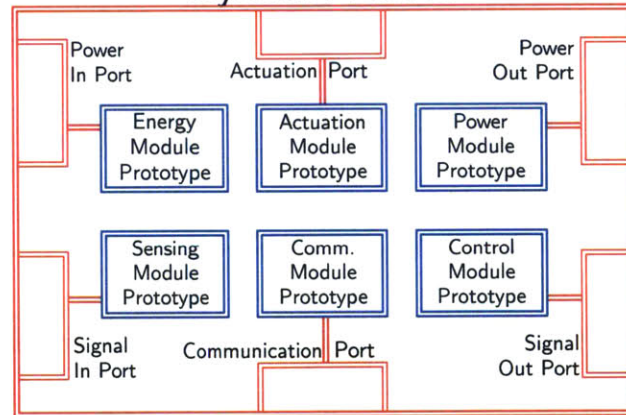
Name: **Electrical System Prototype v0.1**
Parent: Robot Prototype v1.0

Figure 2-4: There are two connection types (signal and power), and three directions (in, out, bidirectional), meaning that there are six ports into the Electrical System Prototype.

Now that the ports of the Electrical System are defined, the modules contained within the system must be enumerated. A design constraint provided by the Robotics Prototype is that each port within the system may only connect to one module. This constraint ensures the routing of connections to one unique module once the connection has entered the system. It follows that there are six modules contained within the electrical system. The modules in the power path are the *Energy Module* (power in), the *Actuation Module* (bidirectional power), and the *Power Module* (Power Out). The signal path modules are the *Sensing Unit* (signal in), the *Communication Module* (bidirectional signal), and the *Control Module* (signal out). The prototypes of the electrical modules are shown in Figure 2-5, along with the connection prototypes connecting the modules to their respective ports. These connection prototypes indicate that the Energy Module is the *only* module that may connect to the Power In Port. Once a connection is inside the Electrical System, there are no restrictions on which modules the signal can be routed to. For example, if a connection of type power-in came into the electrical system, it must go through the Power In Port, then the En-

ergy Module, but after that the signal could reemerge from the Energy module and be routed to any, or all, of the five other modules. By similar logic, if the connection of type power needed to be routed out of the system, the connection would have to go into the Power Module and then be routed out of the Power Out Port.

Robot Prototype - Electrical Domain System Level



Name: **Electrical System Prototype v1.0**

Parent: Robot Prototype v1.0

Figure 2-5: Each of the six system ports has a corresponding module. Any connection entering the Electrical System must be routed to the appropriate module based on the connection's type and direction.

2.2.2 Electrical Modules

At an intuitive level, each module has a role in fulfilling the functionality of the robot. The Energy Module is responsible for housing the energy storage device, e.g. the battery for the robot, and supplying unregulated power to the rest of the modules. A power-in connection to the robot's electrical system would be routed to the Energy Module, where it would most likely recharge the energy storage device.

Internal to the electrical system, the Power Module is responsible for taking the unregulated power output from the Energy Module and regulating the connection to a usable level for the robot. This regulated power could then be routed to all of the other modules. If a robot had a electrical power output connection to the outside

world, the connection must be coming from the Power Module.

The Actuation Module is responsible for supplying bidirectional power to a robot. It is in this module that the robot actuators would reside. In order to effectively drive the actuators, the Actuation Module must be able to both source and sink power, hence the bidirectionality of the connection.

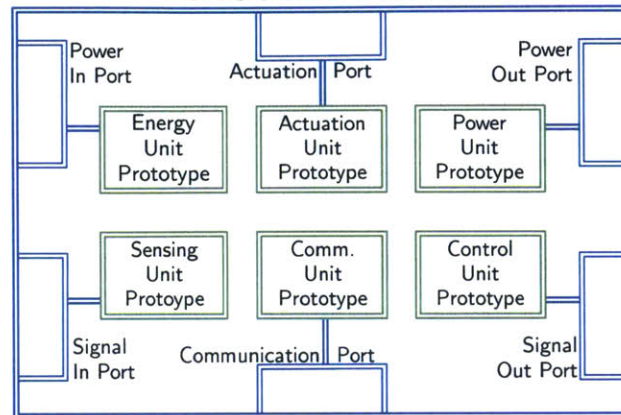
In the signal path, the Sensing Module receives all of the signal inputs. All of the robot's sensors should reside in the sensing module, regardless of whether they are sensing electrical signals or signals from different domains. If the signal originates in a different domain, then the signal would connect to the Sensing Module through a tunnel. The output from the sensor would usually be routed to the Control Module.

The Control Module is responsible for issuing commands to the rest of the electrical system, and ensuring that the robot is behaving nominally. If a robot had an electrical signal output, such as an LED display, the signal must exit the robot's electrical system through the control module.

All messages sent to and from a robot must go through the Communication Module. This includes all radio signals or tethered connections, like USB and Ethernet. The Communication Module takes the information and reports to the Control Module.

Each of the six electrical modules comes from the same Module Prototype. That is, when expanded, all six Module Prototypes look identical. Modules can accept any of the six types of connections (any combination of power or signal and in, out, or bidirectional), implying that, just like the Electrical System, they have six ports. The modules are each composed of six units, resulting in a pattern wherein each System-Module looks the same as the Module-Unit, only at a different level. The six units that make up a module have the same names as the six modules; Energy, Actuation, Power, Sensing, Communication, and Control. Each of the six modules has the same restrictions regarding ports as the modules do. The Power In Port may only connect to the Energy Unit, the Signal Out Port may only connect to the Control Module, etc. Inside a Module the units are all free to connect to one another. The prototype for an Electrical Module is shown in Figure 2-6.

Robot Prototype - Electrical Domain Module Level



Name: **Electrical Module Prototype v1.0**

Parent: Electrical System Prototype v1.0

Figure 2-6: Each module contains six units, one for each type and direction of connection. Units within the same module may be freely connected to each other, regardless of connection type and direction.

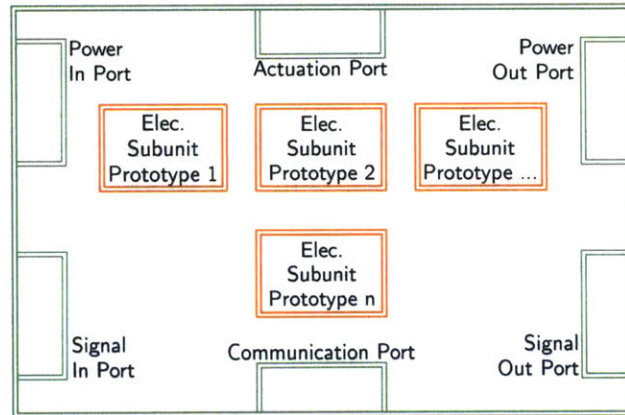
2.2.3 Electrical Units

Each Electrical Module has six Electrical Units, meaning, for example, that the Control Module contains Energy, Actuation, Power, Sensing, Communication, and Control units. This leads to a built-in degree of redundancy that comes with the Robotic Prototype. This redundancy can be utilized for more complex systems, or it can be disregarded for simple systems. The unit that has the same name as its parent module is considered the *critical block* of the parent. The Communication Unit is the critical block of the Communication Module, just like the Energy Unit is the critical block of the Energy Module. More generally, the critical block is defined as the child block of a parent from which the parent block derives its core functionality. Blocks that are not the critical block, but help the critical block achieve its functionality are called *support blocks*. For example, in a robot in Chapter 3, the Energy Unit of the Energy Module (the critical block) contains the battery for the robot, and the Power Unit of the Energy Module (a support block) contains a reverse polarity protection

circuit. In this case, the Energy Unit is the block that the Energy Module derives its core functionality from, but the Power Unit provides additional support.

An expanded electrical unit prototype can be seen in Figure 2-7. Each unit contains the six familiar ports for accepting each of the different types and directions of electrical connections. Inside the unit, there is a departure from the standard six blocks. Each unit is composed of one or more subunits. Because there are an arbitrary number of subunits inside each unit there are no restrictions on which unit ports can connect to any given subunit. For every non-trivial unit, there is one subunit that acts as the critical block for the unit. Other subunits are support blocks.

Robot Prototype - Electrical Domain Unit Level



Name: **Electrical Unit Prototype v1.0**
Parent: Electrical Module Prototype v1.0

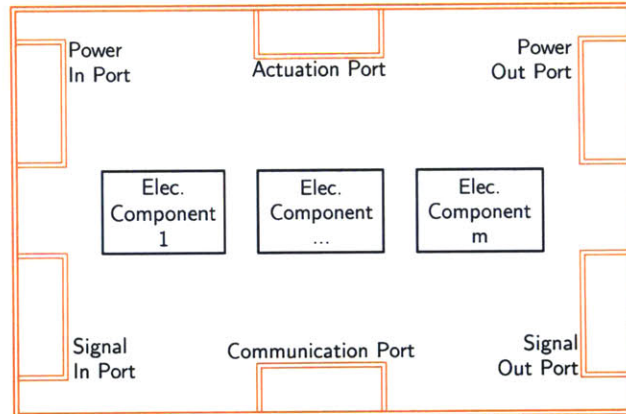
Figure 2-7: As each unit contains an arbitrary number of subunits, there are no prototypical restrictions on which subunits can connect to the unit ports.

2.2.4 Electrical Subunits

Subunits have the six standard ports meaning that, in general, they can accept any type and direction of connections. Each subunit is a group of one or more components that, when given the appropriate inputs, can produce a desired output from the critical component (Figure 2-8). There are no restrictions on which components can connect to the subunit ports. A typical example of a subunit is one whose

critical component is an integrated circuit (IC). The supporting components are the components that come bundled with the IC, such as passives like bypass capacitors and resistors. The subunit must be a fully functional set of components.

Robot Prototype - Electrical Domain Subunit Level



Name: **Electrical Subunit Prototype v1.0**
Parent: Electrical Unit Prototype v1.0

Figure 2-8: Subunits are composed of one or more components. Components are the lowest level block in the Robot Prototype and represent discrete electrical items such as integrated circuits, resistors, capacitors, and inductors.

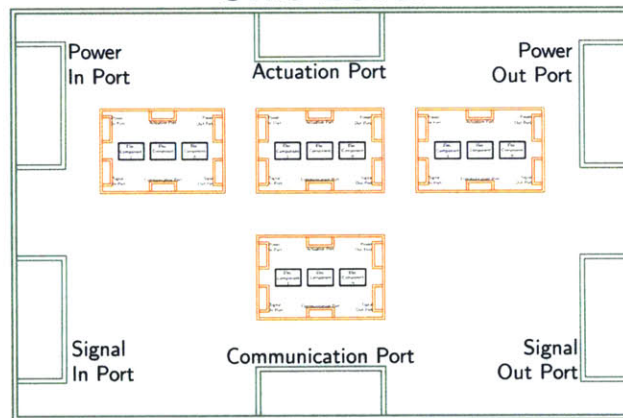
2.2.5 Electrical Components

The component level is the bottom-most level described in the Robot Electrical Prototype. Components are discrete items such as integrated circuits, resistors, capacitors, and inductors. It is worth noting that even though the prototype description stops at the component level, this does not mean that a “subcomponent” level does not exist. System Diagrams can be illustrated to an arbitrarily deep level. Because of the potential for limitless level of depth of the blocks, it is conjectured that System Diagramming can be used to run physics simulations of the systems across multiple domains. These types of multi-domain simulations could be aided tremendously by the Cell Method developed by Tonti [4] [5].

2.2.6 Finalized Electrical Prototypes

By substituting the expanded diagrams for the subunit prototypes back into the unit prototypes, a more complete diagram can be seen in Figure 2-9.

Robot Prototype - Electrical Domain Unit Level



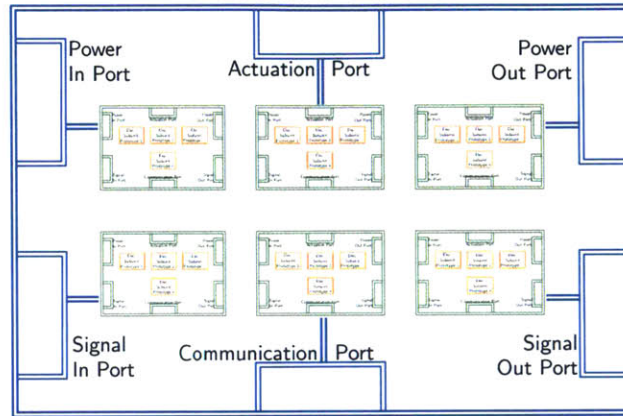
Name: **Electrical Unit Prototype v1.0**

Parent: Electrical Module Prototype v1.0

Figure 2-9: A unit prototype diagram is drawn with expanded subunits.

Inserting the expanded unit prototype diagram of Figure 2-9 back into the diagram for the module prototype yields Figure 2-10.

Robot Prototype - Electrical Domain Module Level

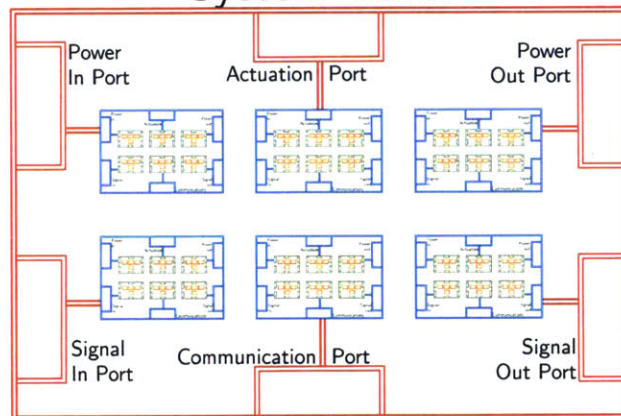


Name: **Electrical Module Prototype v1.0**
Parent: Electrical System Prototype v1.0

Figure 2-10: The result of substituting expanded units into Figure 2-6.

Finally, substituting Figure 2-10 into the system prototype results in the full diagram from the Electrical System Prototype, as seen in Figure 2-11. Because of the sheer number of blocks, drawing a fully expanded system diagram can take up a considerable amount of space.

Robot Prototype - Electrical Domain System Level



Name: **Electrical System Prototype v1.0**
Parent: Robot Prototype v1.0

Figure 2-11: The fully expanded system diagram for the electrical system prototype.

2.3 Mechanical Prototype

Until this point, we have focused on System Diagrams for electrical systems. Now, let us explore systems that have multiple domains. When the rules of System Diagramming were introduced in Chapter 1, connections were defined as the inputs and outputs of blocks, but it was never explicitly stated what those inputs and outputs represented. In the electrical domain, a connection represents two blocks being held at the same potential, with the difference between two potentials being a voltage, and a current traveling through the connections. Voltage and current together represent the power variables of the electrical domain. This idea of power connections being made between blocks is extremely reminiscent of *Bond Graphs* that Paynter introduced in 1959 [3].

2.3.1 Bond Graphs

In Bond Graphs, power factors are separated into two different categories, *Effort* and *Flow* variables. In the Electrical Domain, connections represented electrical potentials and had two types: Signal and Power. The connections represent the electrical Effort variable (voltage), and the item that travels through the connection is the electrical Flow variable (current). In the Mechanical domain, the connection represents the mechanical Effort variable (force) and the item that travels through the connection is the mechanical Through variable (velocity). In the Electrical domain the two types represented a relatively large current (power) and a relatively small current (signal). The distinction between power and signal types are relative to the device that is consuming the current. The current level of an IC is very small compared to the current level of a motor, but in both cases the wires are of type Power if they are delivering energy and not delivering information to the device. Once a System Diagram is constructed, the analogous Bond Graph can be constructed directly from the diagram. This means that the System Diagram can be used not only for designing a system, but also obtaining state equations for a system, which is crucial for simulation techniques.

An example of a bond graph for a simple electrical circuit is shown at the bottom of Figure 2-12, along with the corresponding System Diagram and electrical schematic. In the Bond Graph, SE represents an Effort Source (the battery) and the “1 junction” represents a junction in which the effort variables sum to zero, and the flow variables are all equal (Kirchhoff’s Voltage Law). A “0 junction” represents a junction in which the flow variables sum to zero and the effort variables are all equal (Kirchhoff’s Current Law).

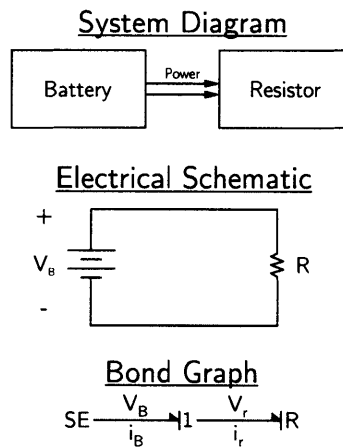


Figure 2-12: The System Diagram, electrical schematic, and Bond Graph for a resistor connected to a battery.

In the Mechanical Domain, wires represent mechanical connections, or forces. The Flow variable is velocity. Similar to the Electrical Domain, there are two types of wires - Static, which represent a relatively small flow (velocity), and Dynamic, which represent a relatively high flow (velocity). Static connections are generally used for structural connections that do not move, while dynamic connections are used for motion coupling. The mechanical and electrical types are compared in Figure 2-13. Note that a static connection does not imply that the force is static, but rather that the block connected to that force does not move. Similarly, in a dynamic connection the force is not necessarily dynamic, “dynamic” implies the the block moves.

Domain	Effort Variable	Flow Variable
Electrical	Voltage	Current
Mechanical (Linear)	Force	Velocity
Mechanical (Angular)	Torque	Angular Velocity

Domain	Relatively Large Flow	Relatively Small Flow
Electrical	Power	Signal
Mechanical	Dynamic	Static

Figure 2-13: Comparison of the different connection types in the Mechanical and Electrical domain

In System Diagramming, moving between domains is achieved with the use of a Tunnel Operator. In Bond Graphs, a tunnel is represented by either a *transformer* or a *gyrator*. A transformer relates the flow variable in the first domain to the flow variable of the second domain (and effort to effort), whereas a gyrator relates the flow variable in the first domain to the effort variable in the second domain (and effort to flow). In Bond Graphs, an electric motor is a gyrator, as it produces a torque (mechanical across) that is proportional to the current (electrical through) running through it, and a back EMF (electrical across) proportional to the velocity (mechanical through). The System Diagram, schematic and Bond Graph of an ideal DC motor driven by a current source and attached to a frictionless wheel is shown in Figure 2-14.

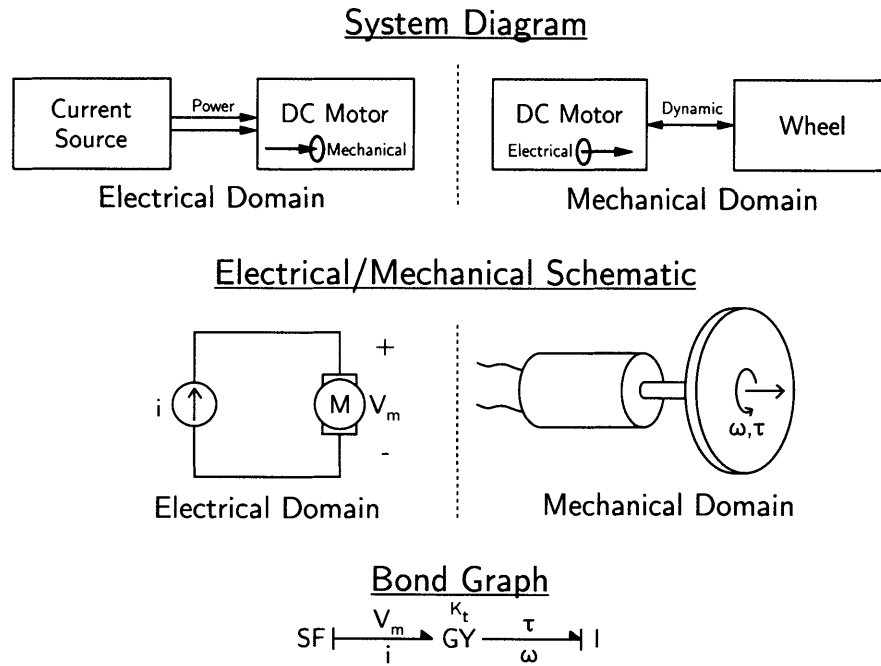


Figure 2-14: The System Diagram, electrical schematic, and Bond Graph for a current source driving a DC motor attached to a frictionless wheel.

Figure 2-14 is an ideal case, without any parasitic components. A more realistic scenario is shown in Figure 2-15 where the inductance, electrical resistance, and friction of the DC motor are accounted for, and the motor is driven by a battery. In both cases the System Diagram stays the same as the parasitics are accounted for inside the DC motor block.

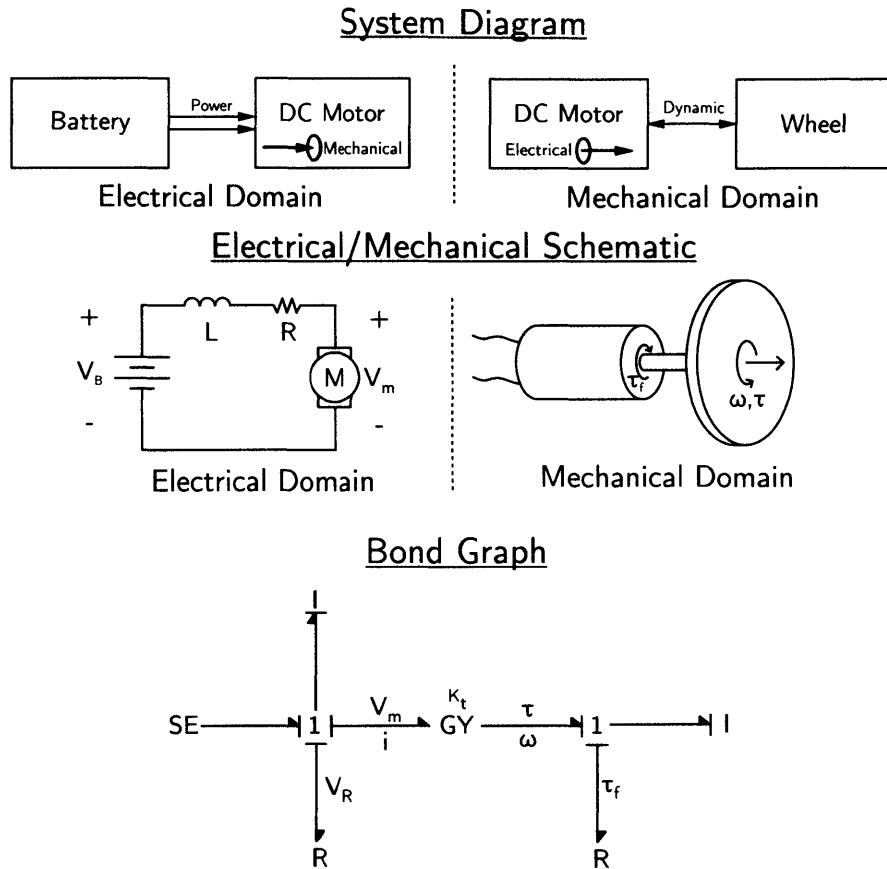


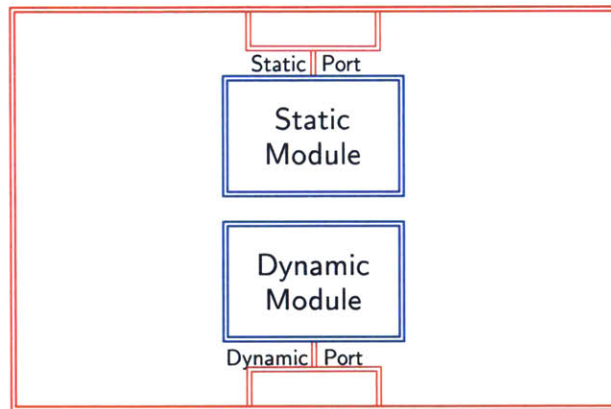
Figure 2-15: The System Diagram, electrical schematic, and Bond Graph for a battery driving a non-ideal DC motor attached to a wheel with friction.

2.3.2 Mechanical System

The Mechanical system prototype has the same hierarchical structure as the electrical system: System, Modules, Units, Subunits, and Components (Figure 2-3). In the electrical case there were six modules, which comes from the fact that there are two electrical connection types (signal and power) and three directionalities (in, out, bidirectional) resulting in six combinations. In the Mechanical System, there are two connection types (Static and Dynamic), but there is only one directionality (bidirectional), which can be seen as an extension of Newton's Second Law. As a result, there are only two Modules contained in the Mechanical system, the Static Module and the Dynamic Module, as seen in Figure 2-16. Only the Static Unit can connect

to the Static Port, and only the Dynamic Unit can connect to the Dynamic Port.

Robot Prototype - Mechanical Domain System Level



Name: **Mechanical System Prototype v1.0**

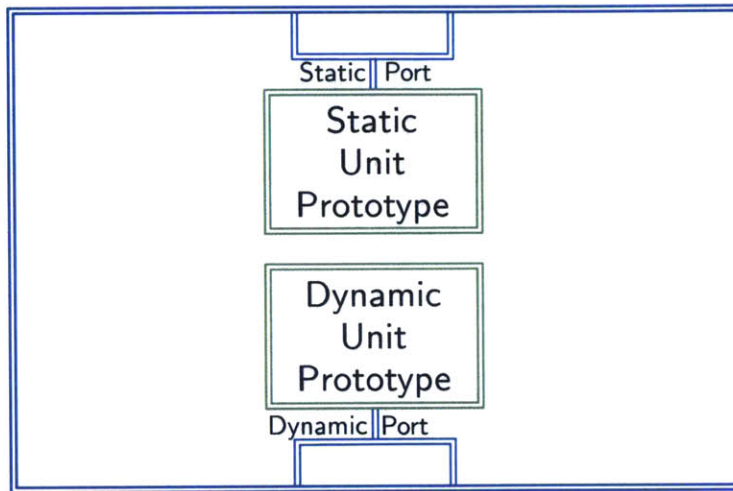
Parent: Robot Prototype v1.0

Figure 2-16: The System prototype for mechanical systems. Only bidirectional connections are allowed in the mechanical domain, meaning that there are fewer ports and modules than in the Electrical domain.

2.3.3 Mechanical Modules

The two mechanical modules can be described as follows - the parts of the mechanical system whose primary purpose is to move, such as the actuators and drive train, belong in the Dynamic Module. The parts of the mechanical system whose primary purpose is to provide structural support, such as a chassis or frame, belong in the Static Module. The two modules can have both static and dynamic connections between them. Each module is comprised of two subunits: the static unit and the dynamic unit, shown in Figure 2-17.

Robot Prototype - Mechanical Domain Module Level



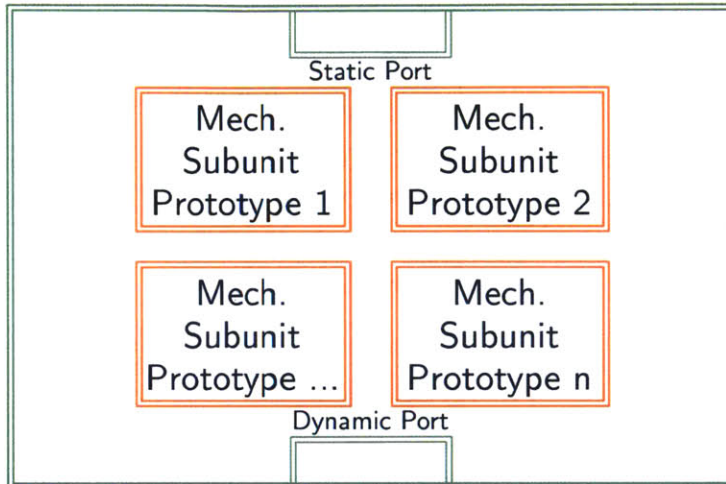
Name: **Mechanical Module Prototype v1.0**
Parent: Mechanical System Prototype v1.0

Figure 2-17: In the Mechanical Module Prototype, only the Dynamic Unit can connect to the Dynamic Port, and only the Static Unit can connect to the static port.

2.3.4 Mechanical Units

The roles for the mechanical units are very similar to the roles that the modules play in the system. The Dynamic Module's critical block, the Dynamic Unit, contains the subunits that move, allowing the Dynamic Module to ultimately achieve its goal. Located in the Dynamic Module's support block, the Static Unit, are the subunits that are stationary, but support the moving subunits. For example, a robot's motor would fall in the Dynamic Unit of the Dynamic module. However, the motor leads that power the motor, but do not move, belong in the Dynamic Module's Static Unit as support blocks. Each mechanical unit is composed of one or more mechanical subunits, as seen in Figure 2-18. There are no restrictions on which subunits may connect to either of the two ports.

Robot Prototype - Mechanical Domain Unit Level



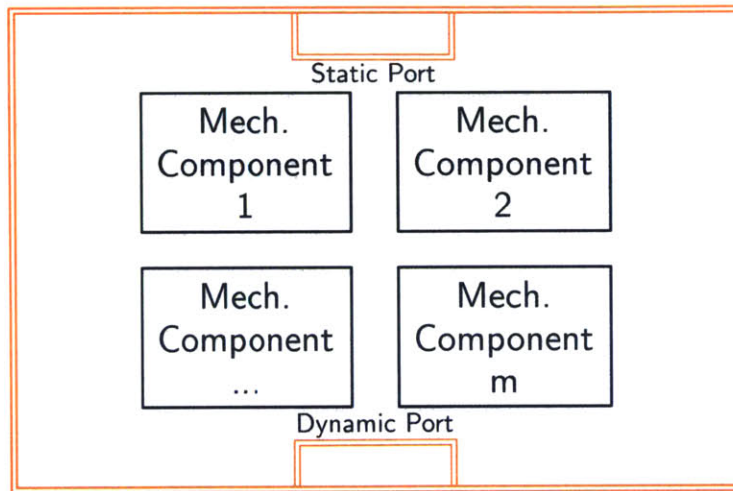
Name: **Mechanical Unit Prototype v1.0**
Parent: Mechanical Module Prototype v1.0

Figure 2-18: The Mechanical Unit Prototype is composed of an arbitrary number of subunits. There are no restrictions on which subunits can connect the Unit's ports.

2.3.5 Mechanical Subunits

Subunits are a group of one or more components that collectively achieve a function, as shown in Figure 2-19. There is a certain degree of freedom in choosing which components belong to which subunit, but components such as nuts, bolts, and machined parts that end up mating together should generally reside in the same subunit. There are no restrictions on which components may connect to the ports of a subunit.

Robot Prototype - Mechanical Domain Subunit Level



Name: **Mechanical Subunit Prototype v1.0**
Parent: Mechanical Module Prototype v1.0

Figure 2-19: The lowest level in the Mechanical System Prototype is the component level. An arbitrary number of components make up the Mechanical Subunit Prototype.

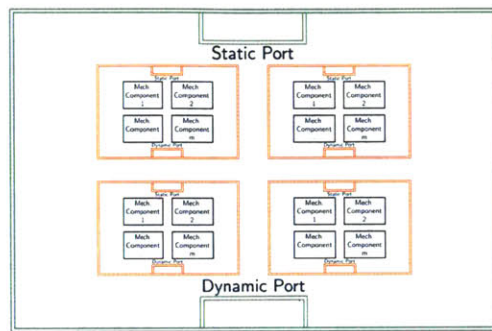
2.3.6 Mechanical Components

Components are the lowest level of the Mechanical Domain in the Robot Prototype. Components are discrete items such as bolts, nuts, washers, or machined parts. Even though the prototype stops at the component level, components can be broken down further. For example, in Solidworks, a part would represent a component. The features in the feature tree of Solidworks that defines the part could be the “subcomponents”.

2.3.7 Finalized Mechanical Prototype

By substituting Figure 2-19 back into the figure for the mechanical unit (Figure 2-18), the expanded Mechanical Unit Prototype diagram can be seen (Figure 2-20).

Robot Prototype - Mechanical Domain Unit Level

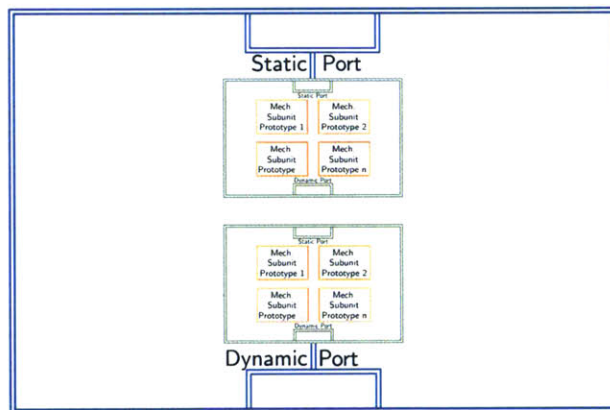


Name: **Mechanical Unit Prototype v1.0**
Parent: Mechanical Module Prototype v1.0

Figure 2-20: Expanding each subunit results in the expanded unit diagram.

Substituting the expanded unit diagram into Figure 2-17 results in the expanded Mechanical Module Prototype diagram, seen in Figure 2-21.

Robot Prototype - Mechanical Domain Module Level

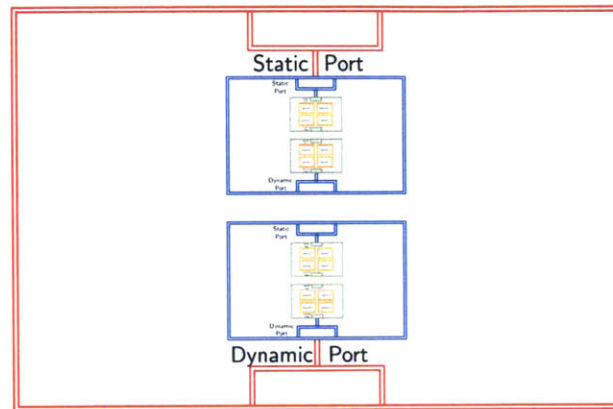


Name: **Mechanical Module Prototype v1.0**
Parent: Mechanical System Prototype v1.0

Figure 2-21: Expanding each unit results in the expanded module diagram.

And finally, substituting Figure 2-21 results in the expanded Mechanical System Prototype, seen in Figure 2-22.

Robot Prototype - Mechanical Domain System Level



Name: **Mechanical System Prototype v1.0**
Parent: Robot Prototype v1.0

Figure 2-22: Expanding each module results in the expanded system diagram.

2.4 Software Prototype

Unlike the Mechanical and Electrical domains, the Robot Software Prototype proposed here is only defined down to the system level. The mechanical system for a robot can be almost entirely independent of the electrical system. However, the same is not true for the software system. In many cases the software system is tightly coupled to the choice of controller in the electronics domain. Furthermore, even if the same controller is used in different robots, multiple computer languages can be used for the software. While this can be accounted for in the software system diagram, it is overly burdensome to force all of the different variations into a single prototype.

In the software domain, a connection represents the passing of data, either through a function call, or a variable assignment. The type of the connection is the data type, e.g. int, float, void, etc. The tremendous number of data types is another reason why fitting a software diagram into a prototype is difficult.

Chapter 3

Modeling a Robot from the System Diagramming Prototype

Representing a robot through its System Diagramming Prototype allows designers to standardize their designs and organize the layout of a robot, regardless of the robot's complexity. This can be a valuable tool for students first learning about robotics, as well as for larger systems that need to coordinate between various teams. This chapter walks through the basics of the robotic prototype and how it can be used to represent simple systems and robots. More complex systems can then be built using this prototype as a starting point.

3.1 Electrical System for a “Charging Robot”

The Robot Prototype proposed in the previous chapter includes three System Level blocks, in accordance with the definition of a robot. For this first example, we will focus on a robot that only has an Electrical System. The robot's sole purpose will be to charge a battery when it is plugged in, and the robot will be nicknamed “ChargeBot”. Because of the extreme simplicity of ChargeBot, it does not meet the criteria to be called a robot at all, because of its lack of Mechanical and Software Systems. Studying it will be useful nonetheless, as this electrical system can easily be incorporated into a full robot. The empty system diagram prototype of ChargeBot is given in Figure 3-1.

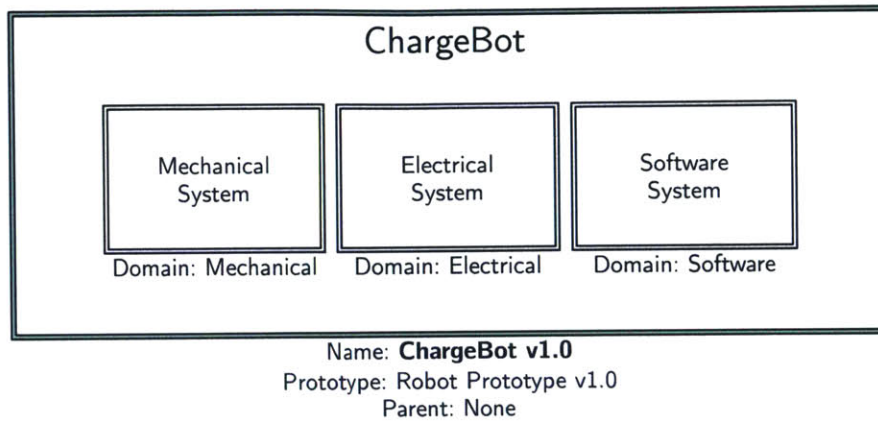


Figure 3-1: Empty System Diagram Prototype for ChargeBot.

To start the System Diagram, we only need to pay attention to the inputs and outputs, regardless of the level of the diagram. In ChargeBot’s case, we will first begin with the Electrical System, the highest level block that needs to be filled in. As stated above, ChargeBot charges a battery when plugged in. Therefore, ChargeBot has two inputs, the charger positive and negative terminals, and no outputs, assuming that the battery is considered part of ChargeBot. Both of the input types are “Power”. The resulting diagram is shown in Figure 3-2.

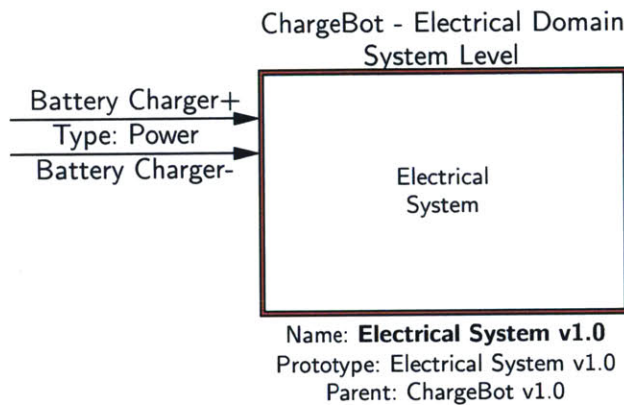


Figure 3-2: Collapsed Electrical System block for ChargeBot.

Now that the system inputs and outputs are defined, we will expand the Electrical System block and look inside it. Creating the system diagram for a robot can be

thought of as the process of following the inputs through the system until they exit the system as outputs, or in ChargeBot's case, terminate inside the system. Once all of the input and outputs have been fully routed, the system diagram is complete.

Because of the type and directionality of the Battery Charger plus and minus inputs (Power, In), the robotic prototype mandates that the connections be routed through the System Power-In Port. From the Power-In port, there is only one possible connection: to the Energy Module. This is shown in Figure 3-3.

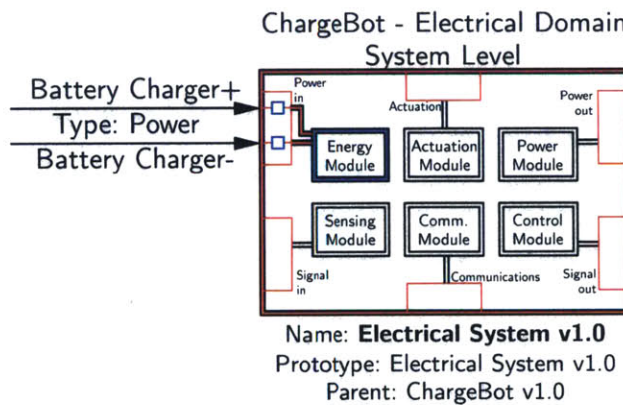


Figure 3-3: Expanded Electrical System of ChargeBot. The battery charger inputs may only be routed to the Energy Module based on their type and directionality.

Following the Battery Charger connections, the inputs must go through the Power-In port of the Energy Module. Once inside the Energy Module, the inputs must then be routed to the Energy Unit as, again, this is all the prototype allows for. ChargeBot's Energy Module is shown in Figure 3-4.

The Robot Prototype does not contain any specific prototype blocks at the Unit level, unlike the System and Module level, each of which contained exactly six blocks - Energy, Actuation, Power, Sensing, Communication, and Controls. We are free to use any subunit we desire to achieve the robot's intended functionality. However, there is still a subunit prototype, but this is mostly to enforce the port conventions that we have been following elsewhere in the Robot Prototype. Based on the initial description of what we want our robot's functionality to be, we know that there must be two subunits - a battery subunit, and a battery charger subunit. The Battery

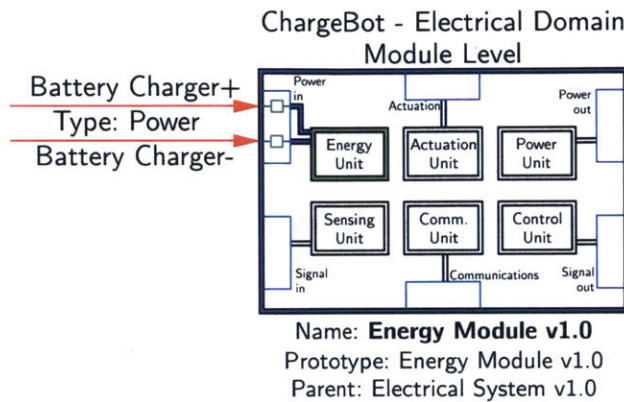


Figure 3-4: Expanded Energy Module block for ChargeBot. The inputs routing to the Energy Unit is mandated by the Robot Prototype.

Charger Subunit, which is responsible for charging the battery, is hooked up to the Battery Subunit, (Figure 3-5). In the simple example of ChargeBot, the battery does not actually supply power to any other device. Of course for any non-trivial robot this would not be the case. In any practical system the Battery Subunit would then be connected to the Energy Unit Power-Out port.

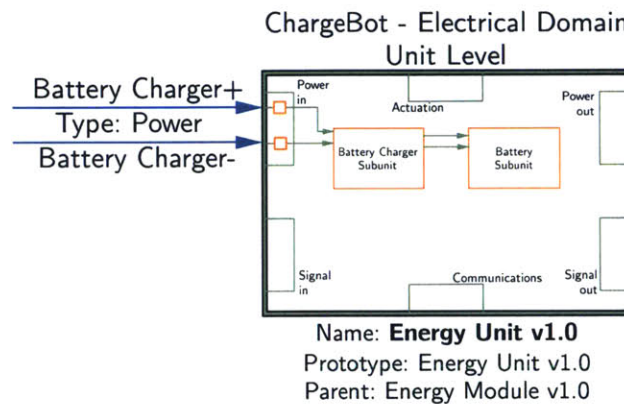


Figure 3-5: Energy Unit of ChargeBot. Note how there are no longer the double lines around the subunits, as there is no specific prototype that the two subunits are filling.

Now that we know where the Battery Charger subunit resides, as well as its inputs and outputs, it is the job of the electrical system designer to source the necessary

components and complete the Battery Charger subunit block. As usual, we start with the inputs and output. The Battery charger inputs must be transformed from a presumable unregulated source to a regulated Battery Charger output connection that goes directly into the battery. Component blocks occupy the level nested inside of subunits. Usually, components would be off-the-shelf (OTS) products such as inductors, resistors, capacitors and integrated circuits, meaning the electrical system designer has little to no control over the components' behavior. The designer does however have complete control over which components to use and how to utilize them. In this case we will design the Battery Charger subunit using, for example, an LTC4065 Lithium-Ion battery charger IC, a 1 μF bypass capacitor, a 2.2 k Ω resistor to set the charging current, and an LED with a current limiting resistor to tell when the battery is charging. This is shown in Figure 3-6 See the LTC4065 data sheet for more information [1]. Given that we have an LED on board now, there is an output from the Battery Charger Subunit that was not previously accounted for. The LED is of type "Signal", and is of direction "Out." We must add this to the system diagram, and we will have to follow the LED output until it exits the system.

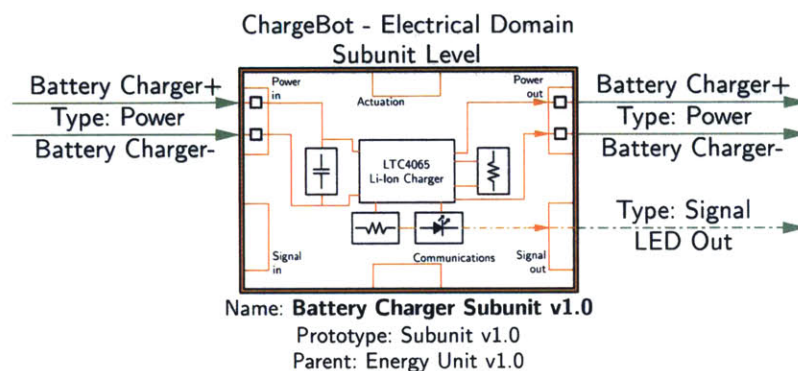


Figure 3-6: The LTC4065 Battery Charger IC is the critical block of the Battery Charging Subunit.

The Battery Subunit (Figure 3-7) contains a battery and two bypass capacitors. Normally the output from the battery would leave through the Power-Out port and power the rest of the robot, but not in the case of ChargeBot.

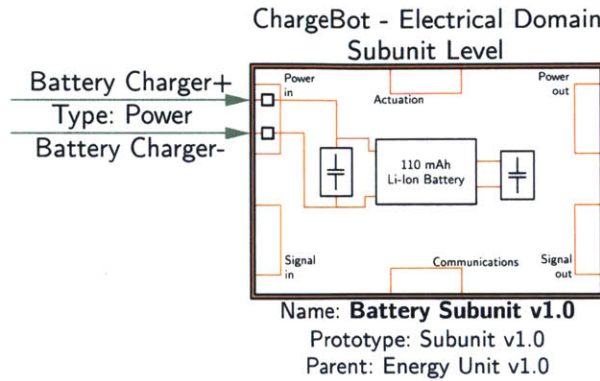


Figure 3-7: The battery subunit of ChargeBot. Normally the battery would have an output to power the rest of the system.

At this point, we will leave the subunit level and return to the Energy Unit. Because there was an output from the Battery Charger Subunit, the LED output, that was not previously accounted for, it is necessary to go back and update the diagram for the Energy Unit (Figure 3-5). The LED signal needs to exit the robotic system and be received by our eye, so we have to route the signal in a manner that exits the system diagram. To begin, the LED signal leaves the Energy Unit through the Signal-Out port, as seen in Figure 3-8.

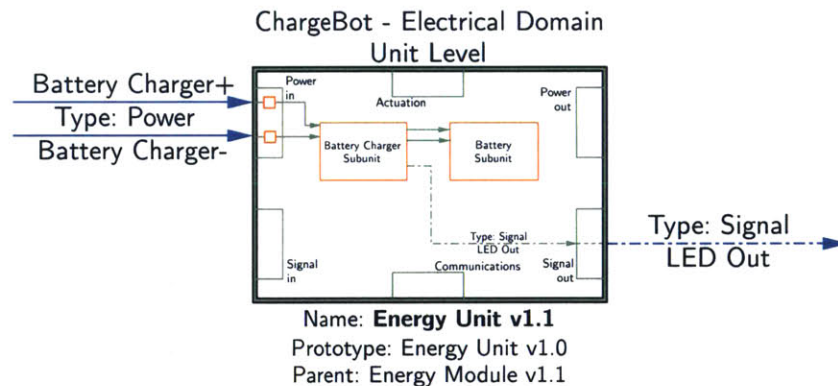


Figure 3-8: Updated version of the Energy Unit (v1.1), now reflecting the LED output signal.

Leaving the Energy Unit and returning back to the Energy module, the LED out signal must be routed out through the Signal-Out port. However, because of the Robot Prototype, the only connection prototype that connects to the Signal-Out port is from the Control Unit. Therefore there is no choice but to route the LED signal to the Control Unit, before it can leave the Energy Module, as seen in Figure 3-9. The constraints imposed by the prototype prove to be useful in larger systems, as it provides for a more formulaic design that results in more consistent results.

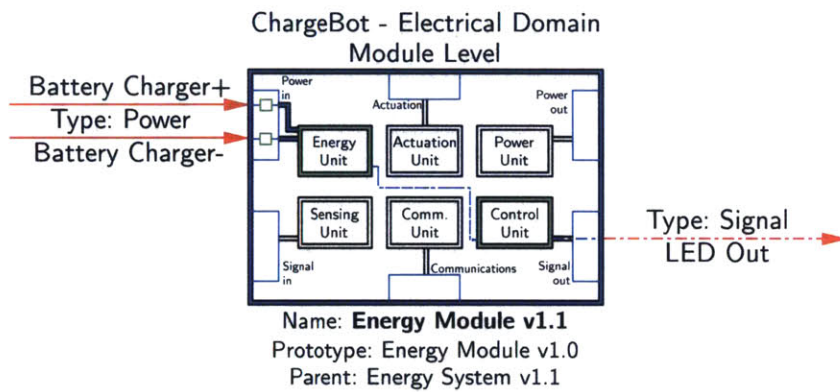


Figure 3-9: LED output must pass through the Control Unit before it can exit through the Signal-Out port.

After expanding the Control Unit of the Energy Module, all that needs to be done is route the LED Out Signal from the Signal-in port to the Signal-Out port, so LED Out can exit through the Energy Module Signal-out port. There are no subunits that the signal has to pass through, so a connection can be drawn directly from the Signal-In port to the Signal-Out port, inside the Control Unit. This is what is called a Pass-Through block, a block whose only purpose is to route signals from one port to another port without first interacting with a block. The Control Unit can be seen in Figure 3-10.

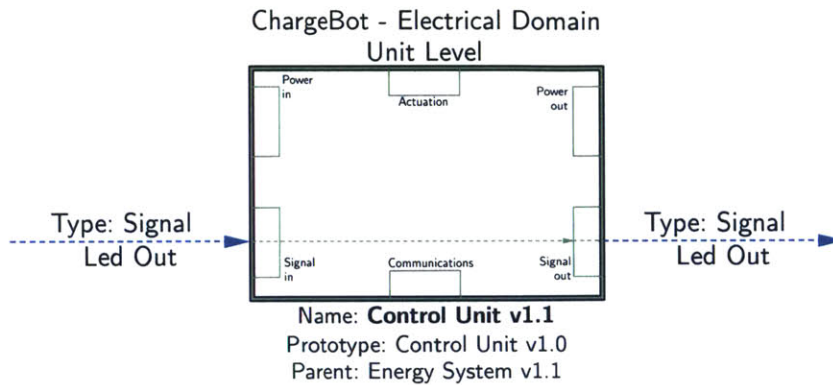


Figure 3-10: A Pass-Through block. The LED signal passes through the Control Unit so that it can be routed out of the Signal-Out Module port.

With the Control Unit of the Energy Module defined, all of the inputs and outputs of the Energy Module are defined and routed accordingly, resulting in the Energy Module being fully defined. We can now back out of the Energy Module and look at the Electrical System. The LED Output connection has to be routed from the Energy Module to the System Signal-Out port. Again, because of the prototype constraint, the signal cannot be routed directly to the port as seen in Figure 3-11, but instead has to go through the Control Module as seen in Figure 3-12.

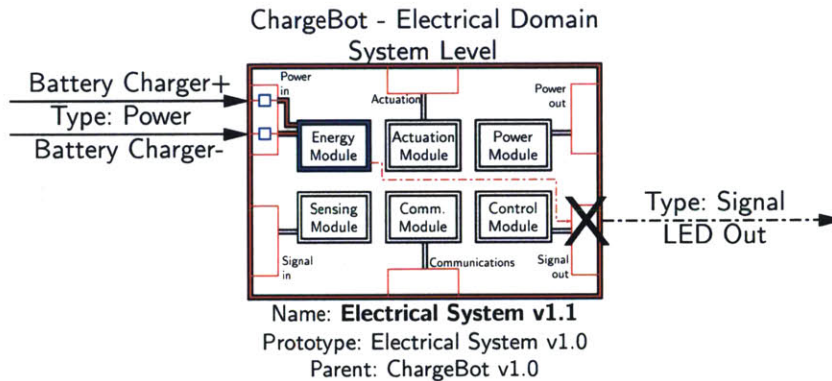


Figure 3-11: Incorrect routing of the LED Output signal. Signals may only exit through the system Signal-Out port from the Control Module.

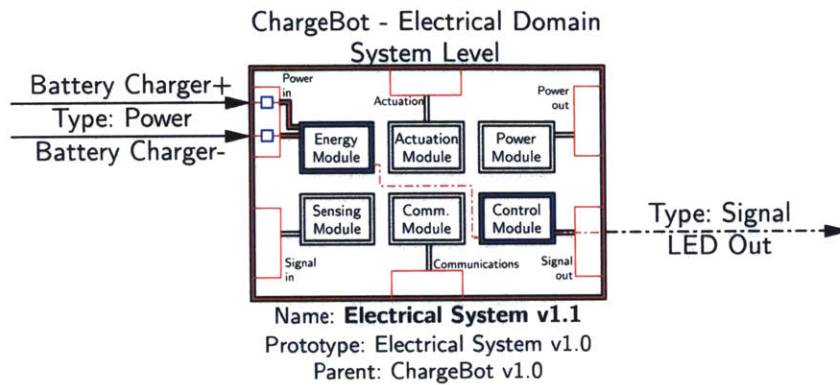


Figure 3-12: Correct routing of the LED Output signal. In this case the Control Module is a pass-through block.

Inside the Control Module, Figure 3-13, the signal must go in through the Sensing Unit, then connect to the Control unit, where it can exit through the Control Module Signal-Out port. There are no prototypical restrictions on which units can connect to other units inside of the same module.

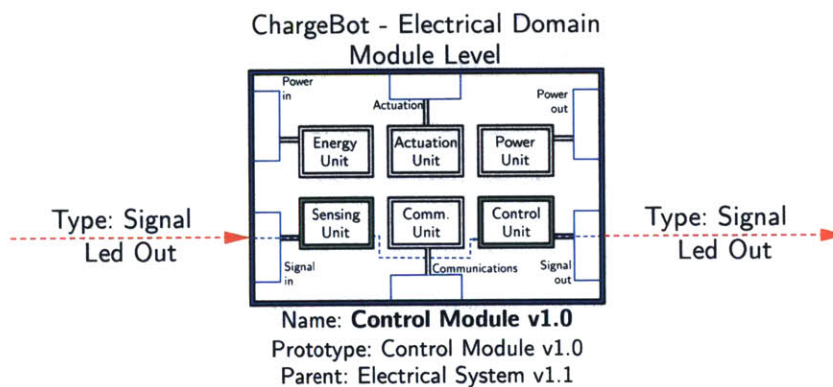


Figure 3-13: LED Out passing through the Control Module in order to exit the system.

This leaves us to define the Control Module's Sensing and Control Units. This is quite easy as both blocks are Pass-Through blocks, as seen in Figures 3-14 and 3-15.

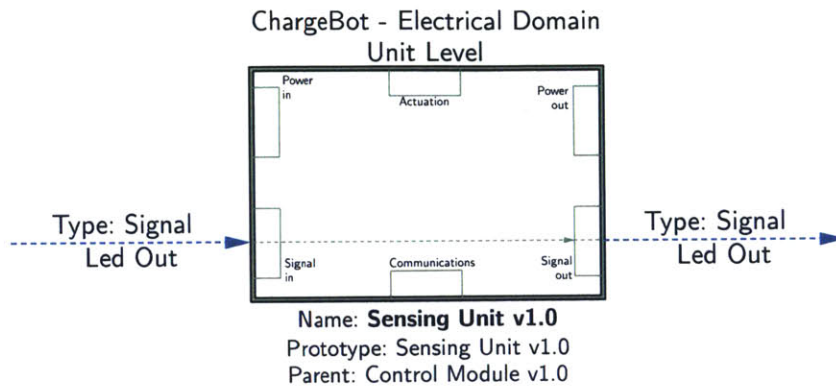


Figure 3-14: Pass-Through Sensing Unit to route out the LED connection.

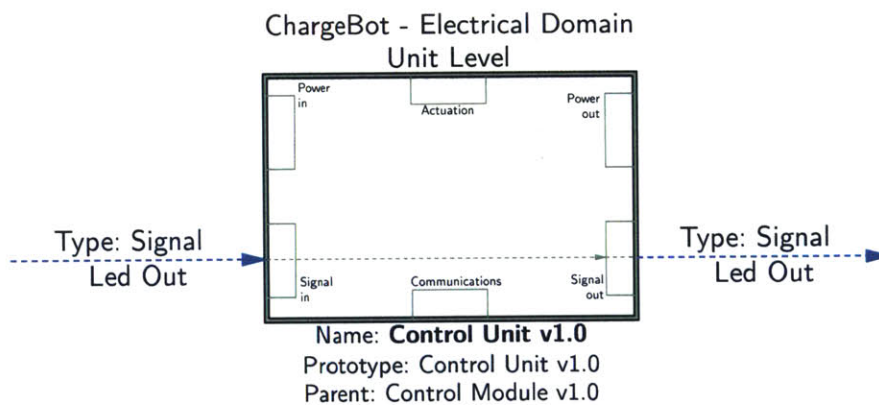


Figure 3-15: Pass-Through Control Unit to route out the LED connection.

With the Control Module's Sensing and Control Units defined, the LED Out signal can successfully leave the system. This means that all of the inputs and outputs are defined, which results in the entire system being defined. The collapsed Electrical system block for ChargeBot can be seen in Figure 3-16. It might seem like a lot of effort went into routing the LED signal out, but this was just to satisfy the prototype constraints of the system diagram, and did not make the physical system any more complex. It is worth noting that there can be many different System Diagrams that all describe the same physical system, that is, each physical system does not have a unique System Diagram.

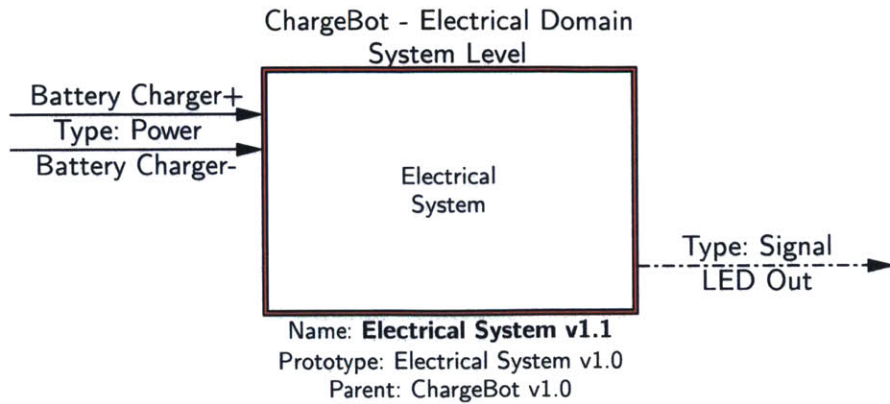


Figure 3-16: Complete Electrical System diagram. Note that this is version V1.1, instead of V1.0 like Figure 3-2

3.2 A Functional Robot Example

In this next section, we will again use the robot prototype proposed in the previous section to create a system diagram for a robot. This robot however, will actually meet the definition of a robot, whereas ChargeBot did not. This simple robot will consist of a chassis, motors, wheels, a battery and a circuit board (Figure 3-17). The robot will be able to drive around based on inputs provided from a mobile device via Bluetooth Low Energy (BLE). The robot will be named “DriveBot.” To begin, we will start with the blank robot prototype in Figure 3-18.

First, a system diagram is drawn without the Robot Prototype (Figure 3-19). At this level, the robot contains the chassis group and two motor groups. Now, let’s fit this system diagram into the Robot Mechanical System Prototype.

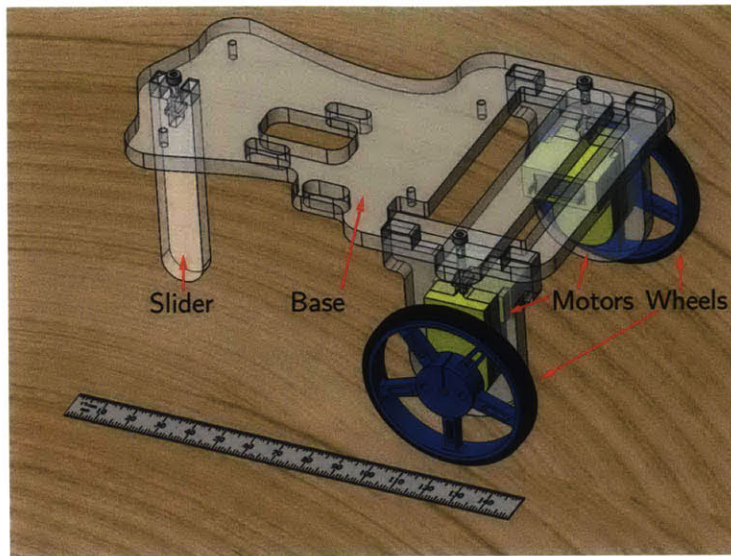


Figure 3-17: A CAD model rendering of the DriveBot.

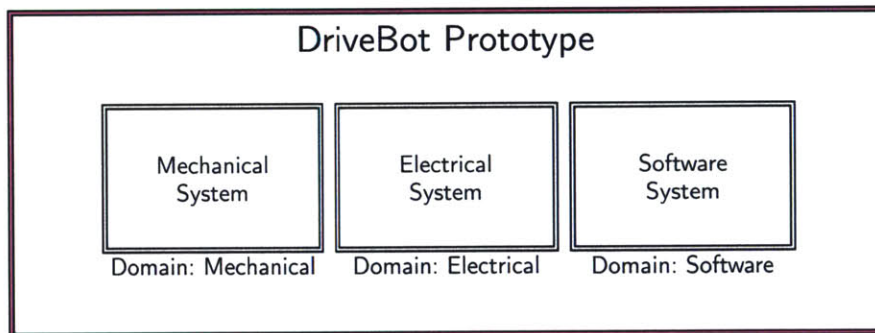


Figure 3-18: Starting point for the DriveBot System Diagram - a blank robotic prototype

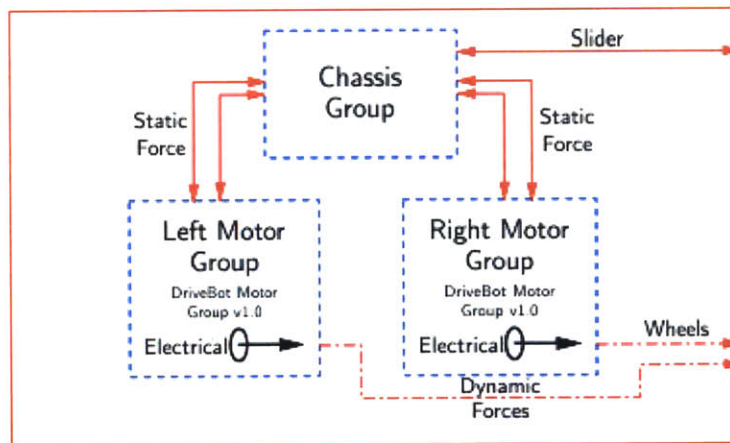


Figure 3-19: A system diagram for DriveBot drawn outside the context of the Robotic Prototype.

We begin by seeing that there are three different forces that the robot exerts on the external world. One occurs at the Slider on the front of the robot, and the other two at each wheel. The Slider never moves relative to the robot, so this force must exit the mechanical system through the static port. Therefore, the Slider must belong to the Static Module. The Static Module prototype will be filled by a block named Chassis Module, version 1.0. In contrast to the Slider, the wheels *do* move relative to the robot, so the forces that the wheels exert on the external system (in this case, a table) must exit through the Dynamic port, meaning that the wheels exist in the Dynamic Module. We populate the Dynamic Module prototype with a block named DC Motor Module, version 1.0. The motors must be held in position to the chassis, so there must be static connections between the DC Motor Modules and the Chassis Module. The Resulting diagram is shown in Figure 3-20.

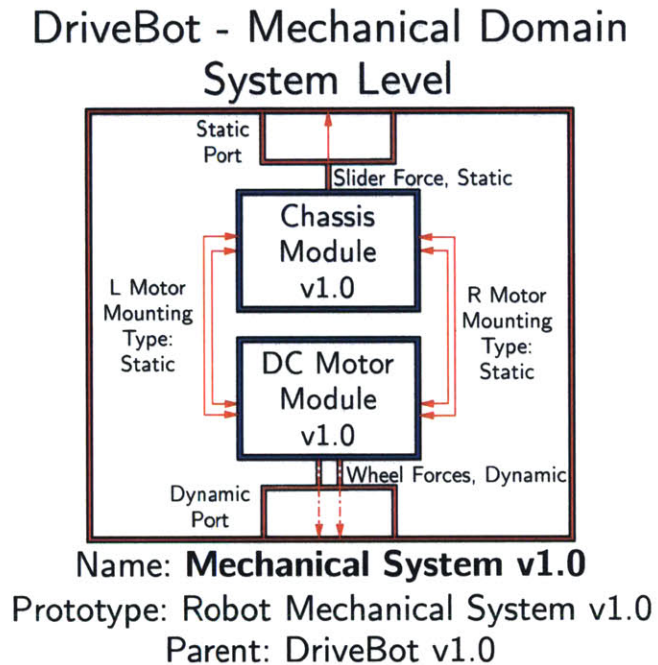


Figure 3-20: A system diagram for DriveBot in the context of the Robotic Prototype.

In general we can show that it takes at least two connections to fix one item relative to another. Take, for example, the mechanism by which the front Slider is connected to the base, shown in detail in Figure 3-21. There are a total of four components

used to connect the Slider to the Base; the two components that we are connecting, and two fasteners. The System Diagram of the connections is shown, independent of any prototypes. Each block has two connections, resulting in a fully fixed system. The top face of the base exerts a force on the head of the bolt, the external thread of the bolt exerts a force on the internal thread of the nut, the top face of the nut exerts a force on the thread holding feature of the Slider, and the Slider exerts a force on the base, closing the loop. The system diagram for this, outside of any prototype context, is shown in Figure 3-22.

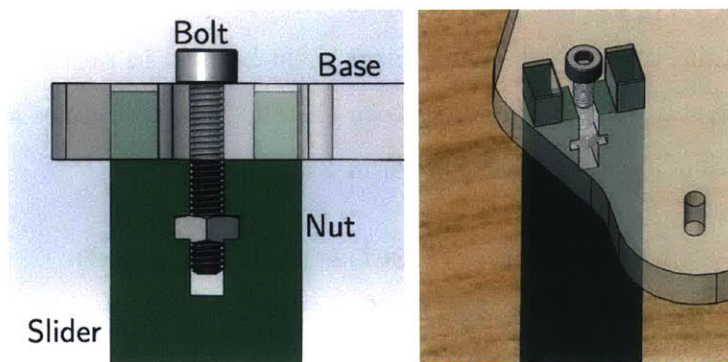


Figure 3-21: A close up of the Slider on the front of the robot.

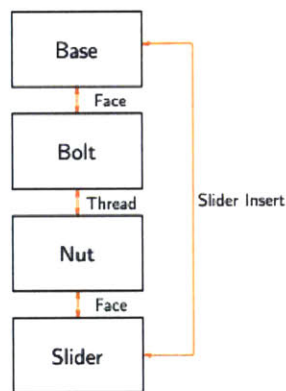


Figure 3-22: System Diagram for the Slider to Chassis connection outside the context of a prototype.

Let's return to the Robot Prototype and make complete the system diagrams for

DriveBot. Diving into the Chassis and DC motor modules, we have several System Diagrams, seen in Figure 3-23. For the Chassis module, nothing moves, so we immediately know that the Dynamic Unit Prototype is left unpopulated. The Dynamic Unit of the Static Module should be viewed as moving parts whose primary role is to support items that should not move. The Static Unit of the Dynamic Module should be viewed as non-moving parts that enable the moving parts to work correctly. The Motors are statically mounted relative to the Chassis of the robot; therefore, the connections from the DC Motor Unit need to go through the Static Unit. There are no components in the Static Unit, meaning that it is a Pass-Through block. The DC motors reside in the Dynamic Unit, as their primary purpose is to output a Dynamic Force. Two forces exit the DC motor Unit, one for each wheel. These forces progress through the entire system, and exit through the mechanical system Dynamic port.

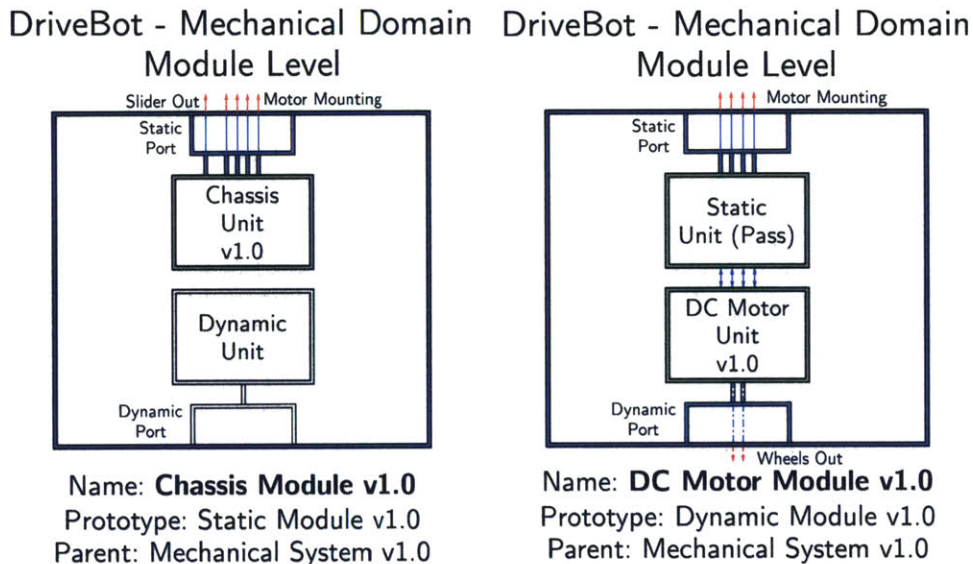


Figure 3-23: The Static and Dynamic Modules of DriveBot, in the context of the robotic framework.

Expanding the Chassis Unit, which occupies the static unit of the Chassis Module, we see the subunits that begin to be recognizable parts of the robot. The Chassis unit is composed of the Chassis, the Slider, and two Motor Upright subunits. The Slider is fixed to the Base with two connections, and each of the uprights contains three

connections - two for fixing the uprights to the Base, and one apiece for securing the Motor Supports, as shown in Figure 3-24. This becomes clear when each subunit is expanded to see the components within.

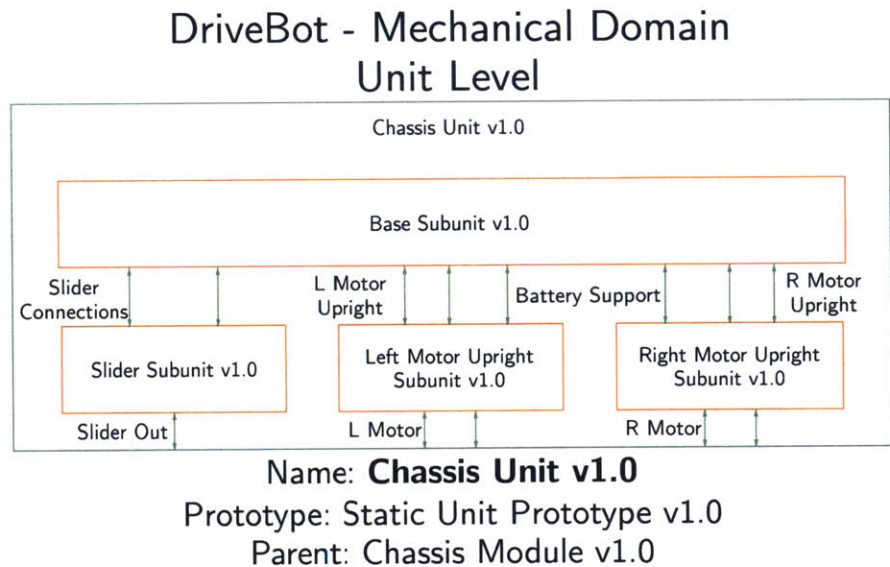
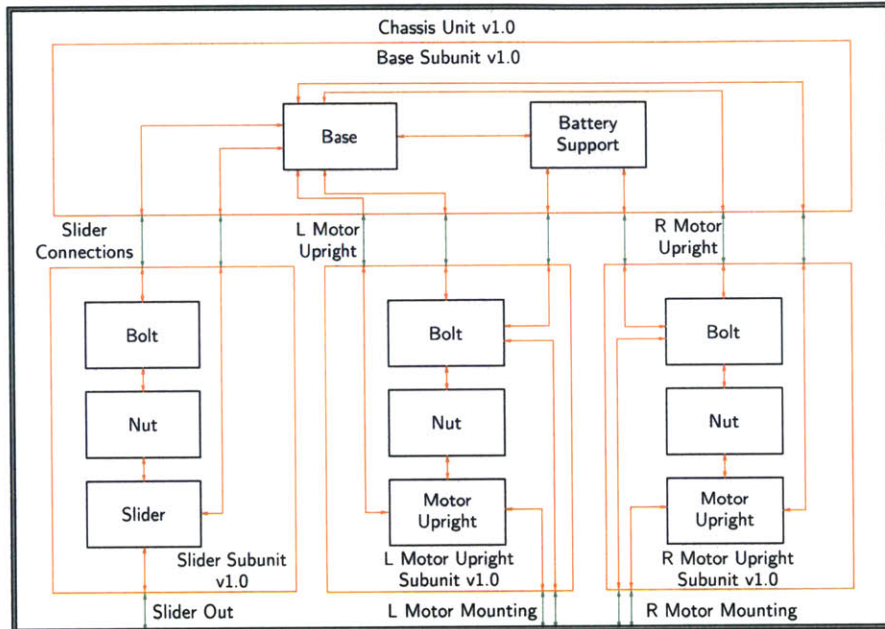


Figure 3-24: System diagram of the Chassis Unit of the Chassis Module for DriveBot.

Once the Chassis subunits have been expanded (Figure 3-25), we can see all of the individual components that make up the chassis. Every component must have at least two connections to be fixed. There is a total of five connections that leave the Chassis Unit; one from the Slider, which will work its way back up the System Diagram and eventually exit through the Robot mechanical System Static port when it comes into contact with the drive surface, and two from each Motor Upright subunit. The two connections that exit from the uprights will go into the DC motor Unit and end up fixing the DC motors to the Chassis.

DriveBot - Mechanical Domain Unit Level



Name: **Chassis Unit v1.0**

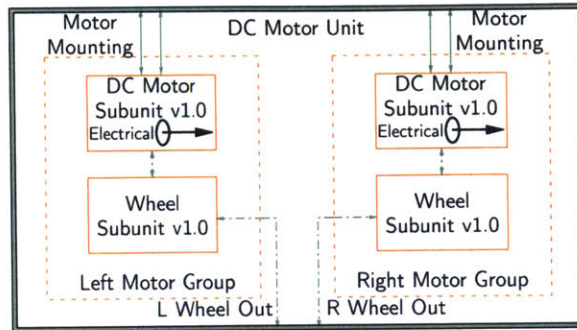
Prototype: Static Unit Prototype v1.0

Parent: Chassis Module v1.0

Figure 3-25: Expanded subunit diagrams for the chassis of DriveBot.

The DC Motor Unit is split into two groups: the left motor group and the right motor group. Each group contains a DC motor subunit and a Wheel subunit. Following the Left Motor (L Motor) and Right Motor (R motor) connections out of the Chassis Unit in Figure 3-26, they go out of the Chassis Module, and into the DC Motors module. There, they go through the Static Module, which is a Pass-Through block, and finally into the DC motor Unit shown on the left. The DC motor subunit contains a tunnel to the electrical domain, which is what enables the dynamic movements. There is only one connection from the DC motor to the Wheel, which implies that the wheel can move relative to the motor, which is true in this configuration, as the wheel is press-fitted onto the motor shaft, and not constrained along the axis of the shaft.

DriveBot - Mechanical Domain Unit Level



Name: **DC Motor Unit v1.0**

Prototype: Dynamic Unit Prototype v1.0

Parent: DC Motor Module v1.0

Figure 3-26: System Diagram for the Dynamic Motor Unit, which occupies the Dynamic Unit of DriveBot mechanical system.

By expanding the DC Motor subunit in Figure 3-27, we can see that the two motor mounting connections are made to the case of the motor. In this simple model of a DC motor, there are six components: the case, the body, two leads, a stator, a rotor, and an output shaft. We know that motors are multi-domain devices so, we expect to see a tunnel inside them, connecting the electrical and mechanical domains. In the case of the motor, there are two tunnels: one on the stator and one on the rotor. The reasoning is as follows: both the stator and the rotor feel a force when an input is provided in the electrical domain (i.e. a current). While the leads of the motor play an integral role in carrying that current, no tunnel exists on the leads because they never experience a force based on an action in the electrical domain (within reasonable bounds). The Lorentz force generated by the current causes the rotor to spin, and as a result, the shaft to spin. The output from the shaft is then transferred to the wheel subunit, where it goes through the wheel body, and out the tire. This force will work its way back up the system and eventually exit through the system dynamic port. In the current system (DriveBot v1.0), there are no actual means to illicit a force from the motors because there is no way to deliver a current to the motors. By looking at the electrical system, this can be immediately discerned,

as the only electrical components are the motors. This is reflected in the mechanical system by seeing that there is nothing hooked up to the leads of the motor.

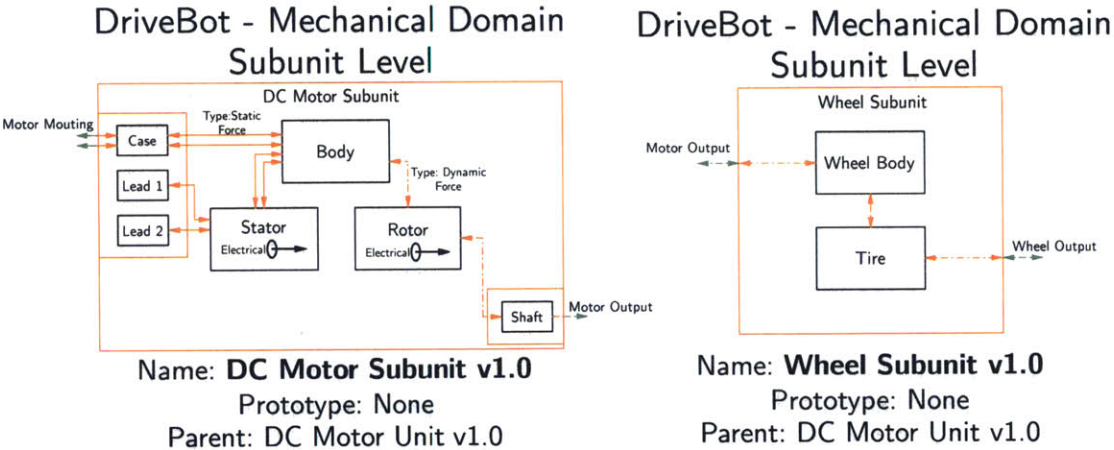


Figure 3-27: A basic motor and wheel diagram for DriveBot.

We will now look more closely at how we can represent the motors, outside the robotic prototype, as in Figure 3-28. In the electrical domain, the connections are straight-forward if we have two components: a battery and a DC motor. All that needs to be done is to connect the two devices together. However, this electrical system has no inputs or outputs. In the mechanical domain, the system is slightly more complex, given that we now need to represent how the battery and motors are connected with a component. In this case we will use alligator clips for the connections. The alligator clips exert a force on the battery terminals and the DC motor terminals. In the mechanical system, when there is an output from the system, the output shaft will turn. This force is being generated from the tunnel connecting the electrical and mechanical domain of the DC motor. The tunnels are drawn here as unidirectional, but in reality they are bidirectional, as moving the DC motor as a system input would generate a current that could charge the battery.

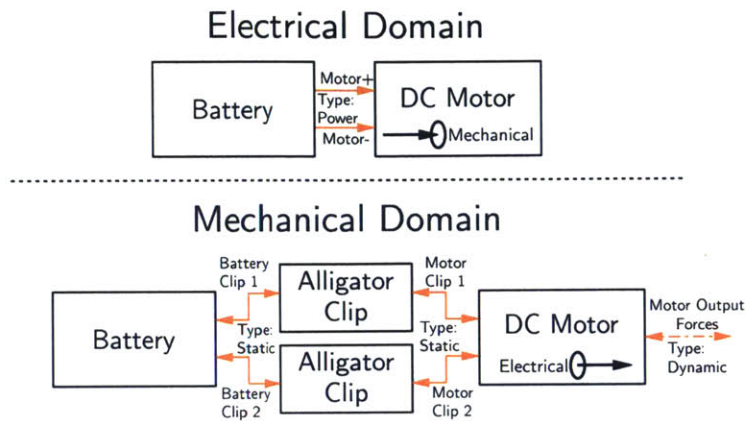


Figure 3-28: A comparison of a DC motor system in the Electrical and Mechanical domains, outside of a prototype.

We want to include a battery and electrical connectors in our System Diagram for DriveBot. First we will show these outside the robot prototype in Figure 3-29. The diagram includes the battery and alligator clip groups. The battery has two connections to each alligator clip group (one per clip), and has a connection to the chassis group (more specifically, the Battery Support component of the Base Subunit belonging to the Chassis Unit.)

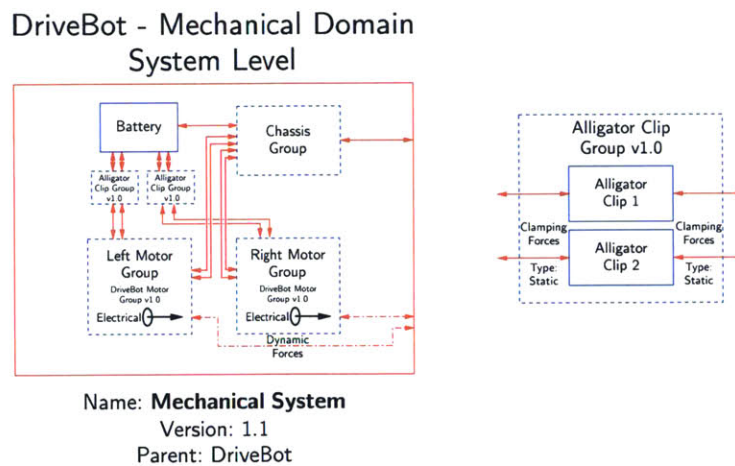


Figure 3-29: Chassis, with battery, motors, and electrical connectors, outside of a prototype context.

Now, we would like to fit the system diagram to our robot prototype. We will

do this by making a new version of the Mechanical system, version 1.1, shown in Figure 3-30. The only difference between the Mechanical System 1.1 and 1.0 is that there is a fifth connection between the chassis module and DC motor module, which is the battery mount. Expanding DC motor Module v1.1, we will replace the Pass-Through block that was occupying the Static Unit prototype, with a new block, called Motor support Unit v1.1. This block will house the Battery and the alligator clips. The reason for putting the battery in the Static Unit of the Dynamic module is that the purpose of the battery is to help the motors accomplish their job of moving the robot around. The motors belong to the Dynamic module, and any items whose purpose is to support other blocks, should reside in the same Module.

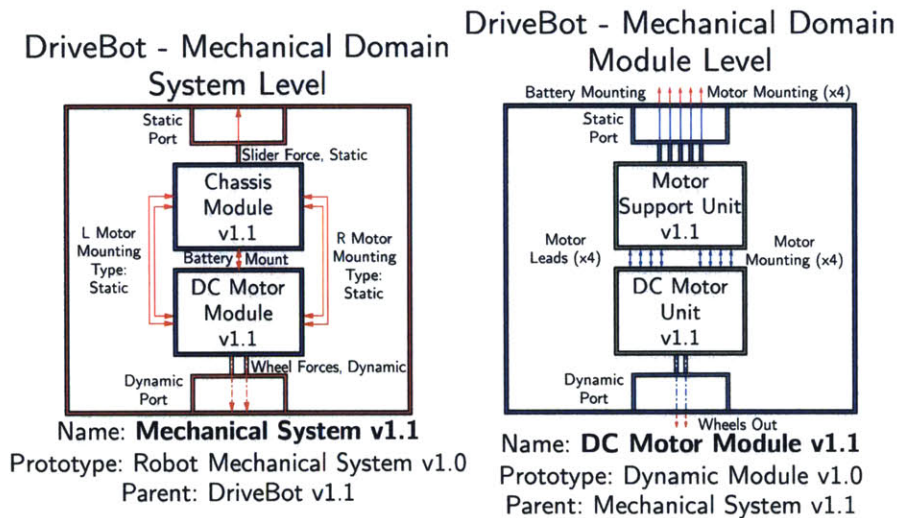


Figure 3-30: The System level mechanical diagram with the DC motor Module expanded.

Looking into the Motor Support Unit v1.1, we see the Battery and Alligator clips. The motor leads connect directly to the DC motor subunits, shown in Figure 3-31.

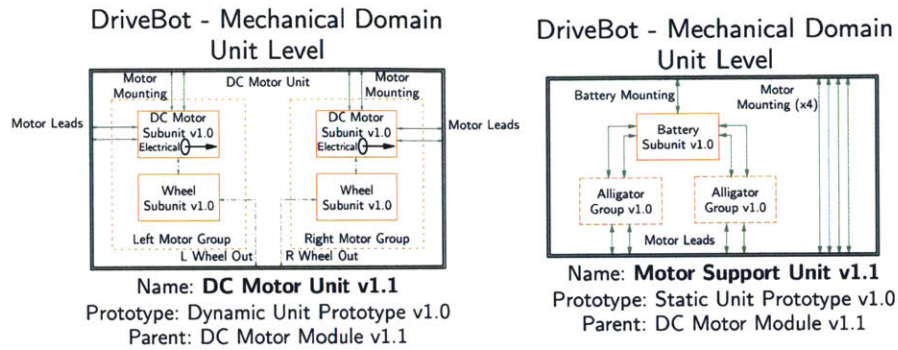


Figure 3-31: The two blocks contained within the DC motor Module. Note the four Pass-Through connections that go through the Static Unit and into the Chassis Module.

This results in the DC motor subunit v1.1 (Figure 3-32). The only difference from v1.0 is that Leads 1 and 2 are connected, whereas they were previously left open. At this point, we have a complete mechanical system for DriveBot v1.1. We will now work on diagramming the electrical system, which at this stage is trivial, consisting of only a battery and two DC motors, Figure 3-33.

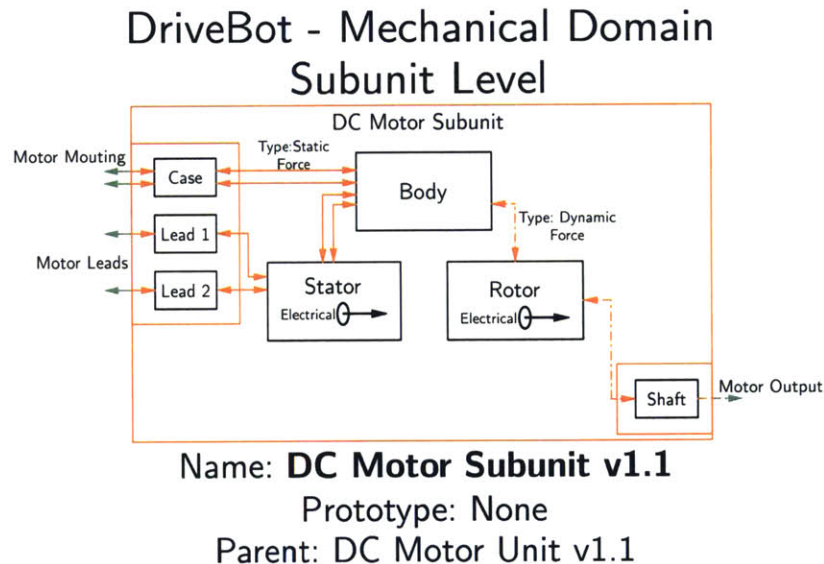


Figure 3-32: DC motor subunit diagram with the motor leads connected to alligator clips.

Electrical Domain

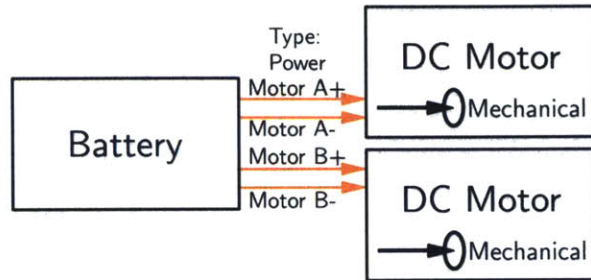
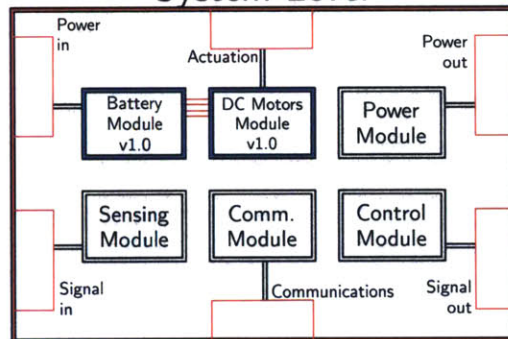


Figure 3-33: The electrical domain system diagram of DriveBot, outside the prototype context.

At the system level, the Battery must go into the Energy module prototype, as the battery is the device that provides energy to the system. Likewise, the DC motor must go into the Actuation module prototype as shown in Figure 3-34.

DriveBot - Electrical Domain System Level



Name: **Electrical System v1.0**
 Prototype: Electrical System v1.0
 Parent: DriveBot v1.1

Figure 3-34: Electrical system diagram for DriveBot, this time in the context of the robot prototype.

Expanding the Battery Module in Figure 3-35, we have the six units of the Energy Module prototype. Because we only have one component (the battery), it is immediately known that the battery must go into the Energy Unit Prototype, the critical block for the module. There are four connections coming out of the battery,

all of type power. These connections are going to the motors, which are in the Actuation Module Prototype, meaning that the connections leaving the battery must exit the Energy Module. No choice is provided to us by the prototype as to how this is achieved; the power connections must exit through the power port, meaning that they must first go into the power unit. In the Actuation Module prototype, the motors must be in the critical block, the actuation unit. The only way to get power into the module is through the Power In port, which only connects to the Energy Unit. Therefore, the Battery to Motor connections must pass through the Energy unit, where they are free to connect to the Actuation Unit prototype.

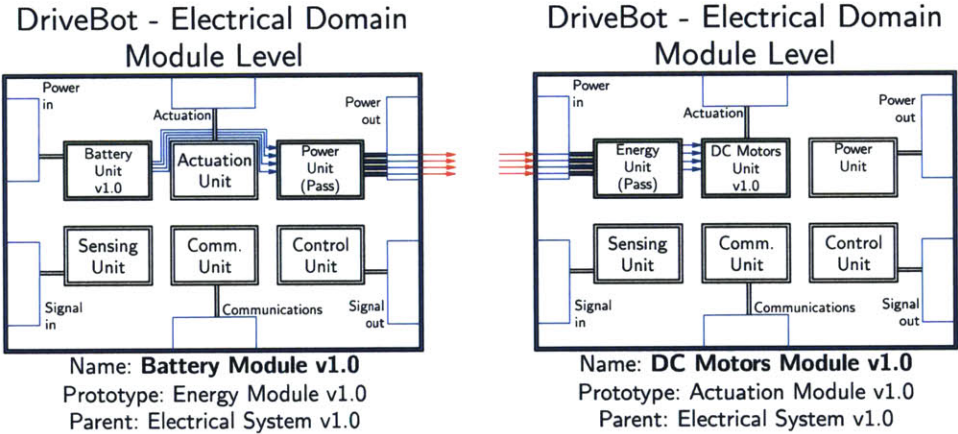


Figure 3-35: The Battery and DC motor electrical modules, inside the context of the robot prototype.

Expanding the only two units that are not Pass-Through blocks, we can see the Battery subunit in the Battery unit, and the DC motor subunits in the DC Motors Unit (Figure 3-36).

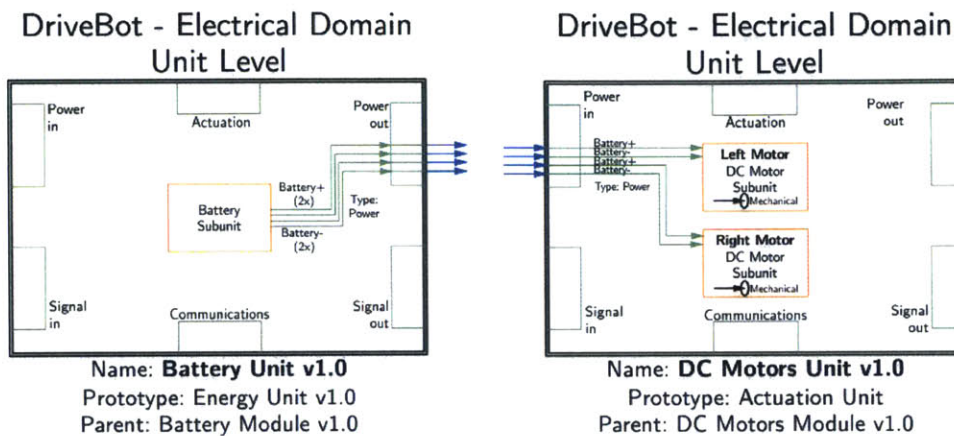


Figure 3-36: The critical electrical units for DriveBot v1.1.

Figure 3-37 shows the finished electrical system diagram for DriveBot v1.1. At this point, we have a complete mechanical system and a complete, although extremely simplistic, electrical system that will allow DriveBot to move over a surface. However, DriveBot is still not a robot by our definition. It does not have controlled output, and it does not have a software system. In order to provide a software system, we must first upgrade our electrical system.

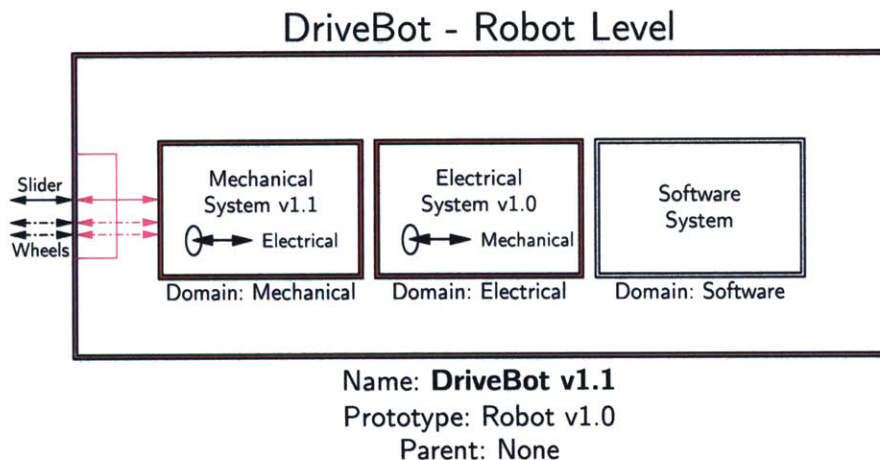


Figure 3-37: Finished System diagram for version 1.1 of DriveBot. Notice that the Software system prototype is still empty, meaning that DriveBot is not yet a Robot. DriveBot will meet the definition for a robot in a version presented later in this chapter.

3.3 Electrical System

To achieve the goal of driving DriveBot around based on inputs from a mobile device, there needs to be a software system. The largest barrier to this in DriveBot v1.1 is the lack of hardware to support software. A microcontroller, or suitable alternative, must be incorporated. To achieve this, a Control Module is added to the electrical system. The Control Module's critical block, the Control Unit, is populated with a microcontroller subunit, as shown in Figure 3-38. The microcontroller subunit will not be expanded here, but it contains all of the necessary passives, filters, and clocks for the microcontroller component to function properly. As indicated in the diagram, we must provide power to the microcontroller subunit, as well as a bidirectional communication path and connections for controlling the motor.

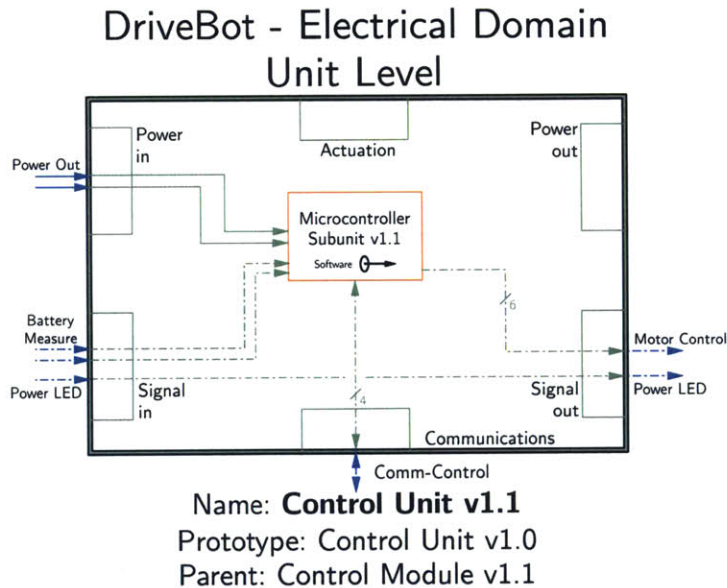


Figure 3-38: A microcontroller is added in the Control Unit of the Control Module.

Moving up a level, the Control Module's role is to provide the necessary inputs and outputs to its critical block, the Control Unit, which was defined in Figure 3-38. This results in the Control Module as shown in Figure 3-39. In this simple robot only the critical block is populated, with only pass-through blocks elsewhere in the module, a theme that will appear again. Now that we have provided a microcontroller, there is

a platform for the Software System, fulfilling a major step of reaching the criteria for a robot.

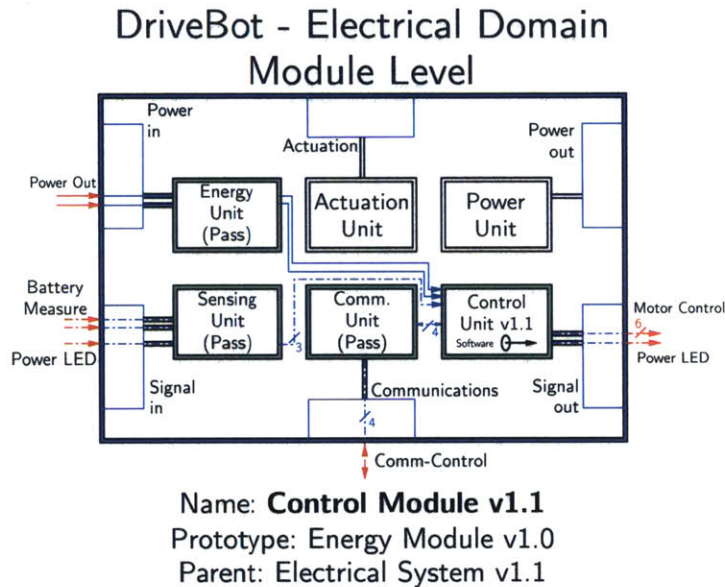
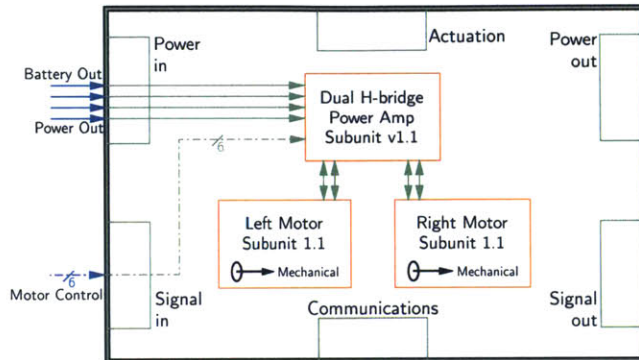


Figure 3-39: The critical block of the Control Module is the Control Unit, which contains a microcontroller. Only Pass-Through units are required elsewhere in the Module.

Using the Control Module as a starting point, there are several options for which blocks to populate next. The previous version of DriveBot was missing two features to be considered a robot. The microcontroller allows for software to be incorporated, but there is still the issue of creating a *controlled* output. In version 1.0, DriveBot would move when the alligator clips were connected to the battery and motors, but there was no way of controlling the velocity of the motors. Adding a microcontroller gives us a platform for creating the signals necessary to control a motor, but the motors take *power* connections, not signal connections. This means that the Actuation block needs to include a block that transforms a signal connection into a power connection, or in other words, we need a power amplifier. The critical Actuation Unit is shown in Figure 3-40 with the power amplifier and two motors.

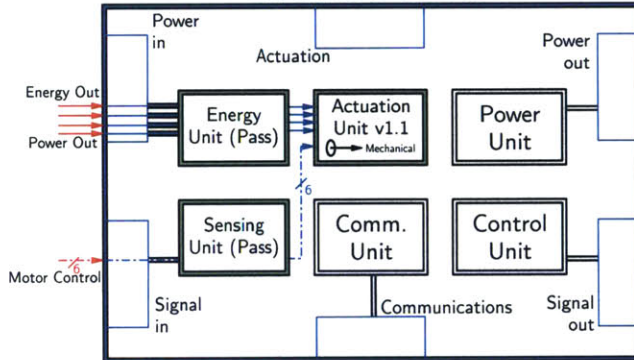
DriveBot - Electrical Domain Unit Level



Name: **Actuation Unit v1.1**
 Prototype: Actuation Unit v1.0
 Parent: Actuation Module v1.1

Figure 3-40: A dual H-bridge power amplifier is used to drive the two motors bidirectionally.

DriveBot - Electrical Domain Module Level



Name: **Actuation Module v1.1**
 Prototype: Energy Module v1.0
 Parent: Electrical System v1.1

Figure 3-41: Two pass-through units are needed to support the critical Actuation Unit.

Following the connections to the Actuation Module in Figure 3-41, there is Motor Control, which came from the Control Module, the Energy Out connections, and the Power Out connections. Power Out is a regulated voltage for logic control, and

Energy Out is an unregulated source that is connected to the main energy source. Following the Power Out connection brings us to the Power Module. Jumping down to the critical block, shown in Figure 3-42, the linear regulator subunit converts the unregulated Energy Out input into the regulated Power Out.

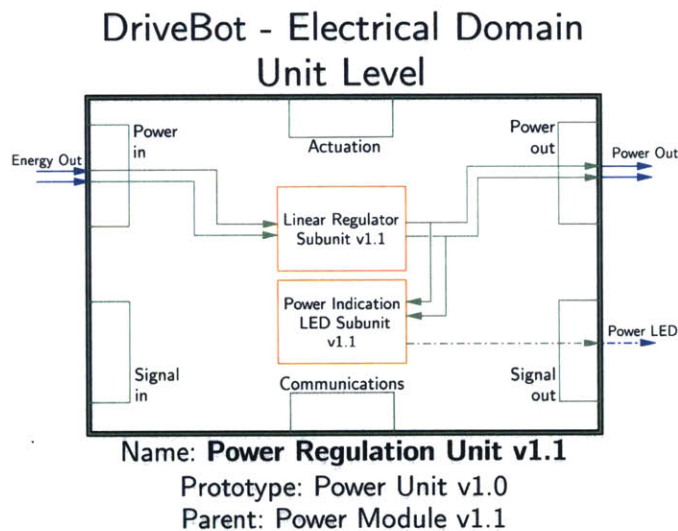


Figure 3-42: The Linear Regulator subunit produces a regulated output from an unregulated input. A power indication LED is used to show when the robot is powered on.

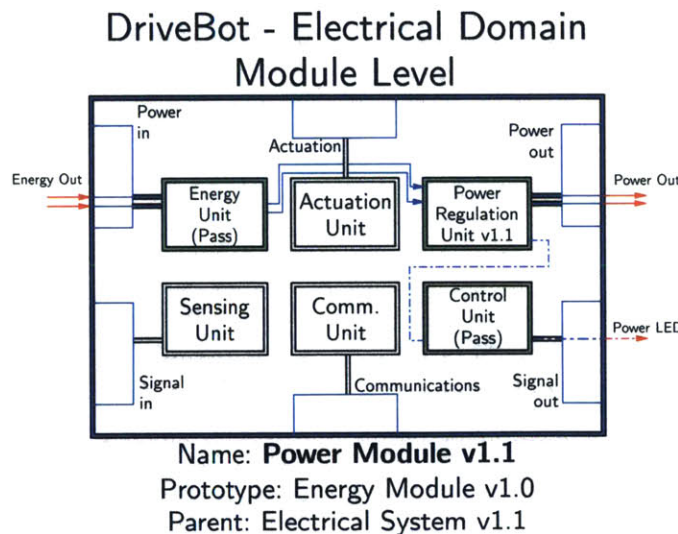


Figure 3-43: The Power Module for DriveBot.

Additionally, there is a second subunit in the critical Power Unit that shows when the robot is on via an LED. The Power LED subunit could have equivalently been put in the Control Unit of the Power Module, which would then need to be a pass-through block. The Power Module diagram is shown in Figure 3-43.

The Energy Out connection has been seen several times now, so let us address its origin. The critical Energy Unit contains the Battery Subunit. It is worth noting that there is an additional subunit in the Energy Unit: the Battery Connector Subunit. This indicates that the battery is not permanently attached to the rest of the electrical system, which is useful if the battery needs to be changed. Figure 3-44 also shows the Energy Module's Power Unit, which is the first time that a non-critical, non-pass-through subunit has been used. This unit provides two very important functions. First, it contains a power switch that allows the user to turn the robot on and off. The mechanical tunnel indicates that this is a mechanical switch, as opposed to a solid state switch, such as a transistor. Secondly, the Power Unit provides reverse polarity protection, which is crucial for any robust electronic system. Inserting the battery in the incorrect orientation should not destroy sensitive electronics downstream. Using the reverse polarity protection subunit in the Energy Module, as opposed to the Power Module, ensures that the Energy Out connections can only be positive, which is safer than doing the protection in the Power Module. The Energy Module, Figure 3-45, contains no pass-through blocks, because we have used the reverse protection circuitry.

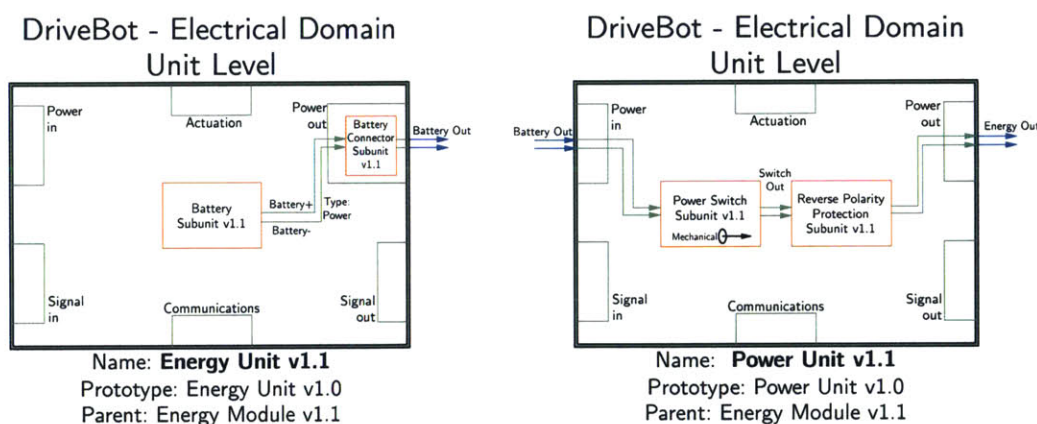
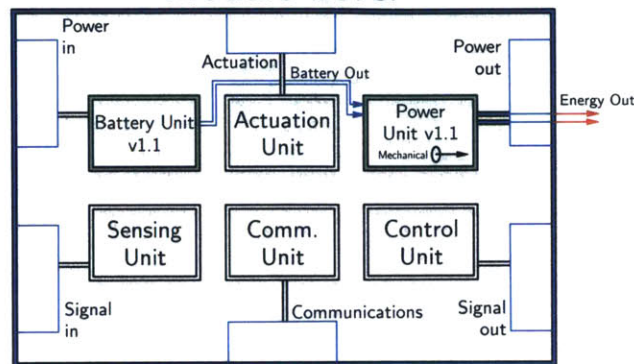


Figure 3-44: The battery, circuit protection, and power switch subunits.

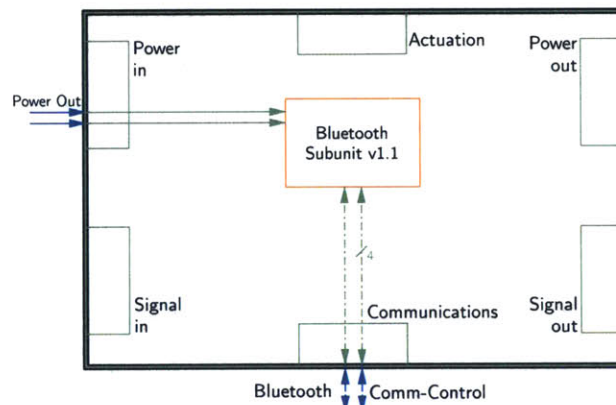
DriveBot - Electrical Domain Module Level



Name: **Energy Module v1.1**
 Prototype: Energy Module v1.0
 Parent: Electrical System v1.1

Figure 3-45: Completed energy module

DriveBot - Electrical Domain Unit Level



Name: **Communication Unit v1.1**
 Prototype: Communication Unit v1.0
 Parent: Communication Module v1.1

Figure 3-46: The communication module enables a user to send commands via a mobile device. The commands are then sent to the control module for execution.

There are two modules that have not been discussed: Sensing and Communication. In the Control Module, there were four bidirectional signal connections that went to the Communications Module, so we will address the Communication block

now. The primary purpose of the Communication Module for DriveBot is to receive commands from a mobile device over Bluetooth. These commands are then sent to the control module where they are acted upon. The Bluetooth Subunit inside of the critical Communication Unit (Figure 3-46) needs a regulated power input, a Bluetooth antenna connection, and the four connections to the Control Module. The resulting module is straightforward, with only one Pass-Through block, Figure 3-47.

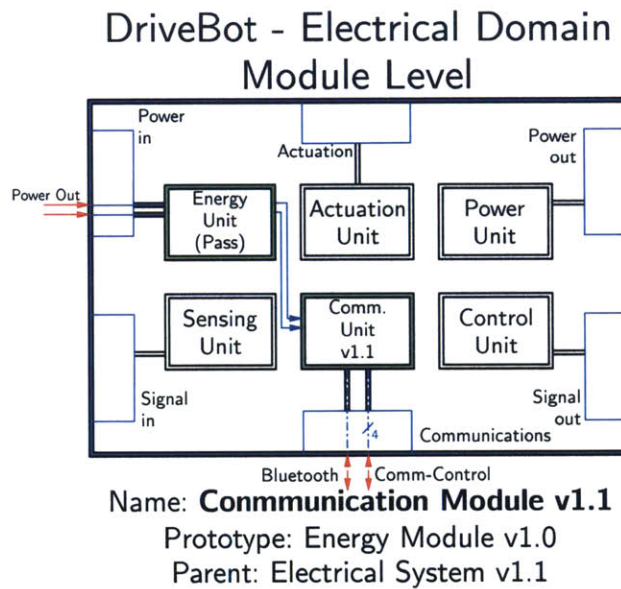


Figure 3-47: Only one pass-through block, supplying power, is needed for the Communication Module.

At this point, the electrical system for DriveBot could be considered complete based on the initial requirements, as it is fully functional. However, we will add a simple sensing unit so that all six modules will be populated. The role of this simple module will be to measure the battery's voltage level. As seen in Figure 3-48, the Battery Measure Subunit takes in the output from the Energy Module, converts it to a signal type, and sends it out to the control unit. The Sensing Module, Figure 3-49, shows that two pass-through blocks are needed.

DriveBot - Electrical Domain Unit Level

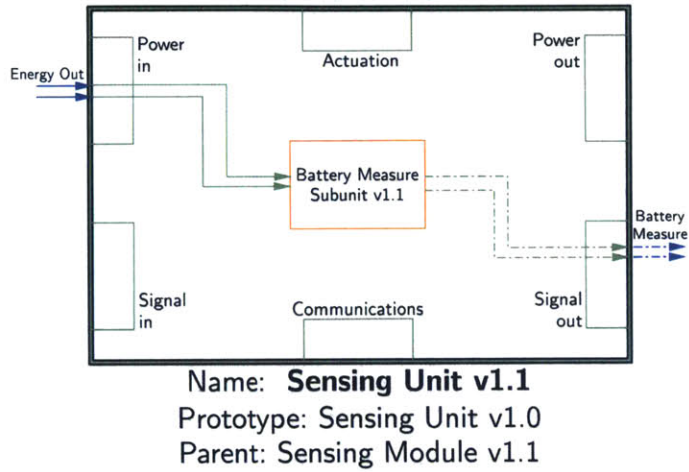


Figure 3-48: The Battery Measure block converts a power type to a signal type.

DriveBot - Electrical Domain Module Level

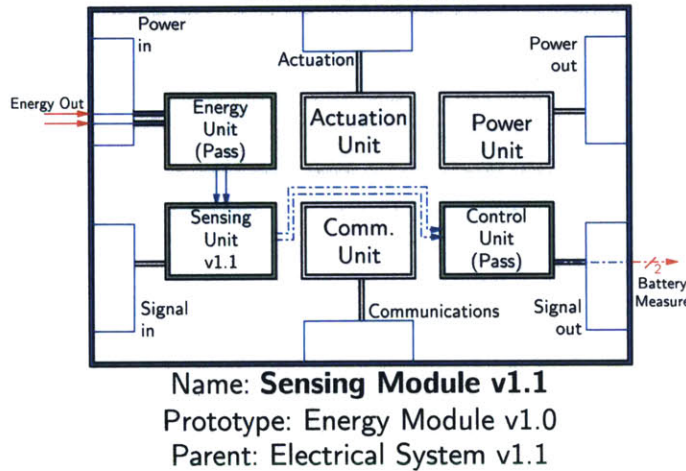


Figure 3-49: Two Pass-Through blocks are required in the Sensing Module.

All of the modules and units have now been diagrammed, meaning that the Electrical System diagram can be completed, as shown in Figure 3-50. There are only two inputs and outputs from the electrical system. One is the Bidirectional Bluetooth connection for receiving commands, and the second is the output from the power in-

dication LED. There is no output from the Actuation Module, as one might expect, because the motors are considered part of the electrical system. The output from the motors comes from the mechanical domain, which is connected to the electrical Actuation Module by way of a tunnel. Also, based on the inputs, it is apparent that there is no way to charge this robot's battery, as the Energy Module does not have any inputs. This functionality could be added by inclusion of the battery charger subunit from ChargeBot's Energy Unit, as presented in Section 3.1.

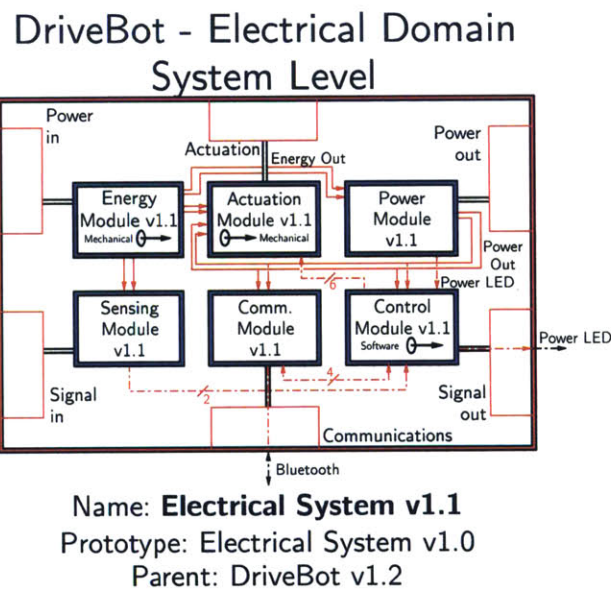


Figure 3-50: Completed Electrical system diagram from DriveBot v1.1

3.4 Software

Now that the electrical system has been completed, there is a platform for the software system. Unlike the electrical and mechanical systems, a prototype for the software system is not proposed. There is immense variability in the architecture of hardware that the software runs, as well as in the languages that are used. These, coupled together with the wide freedom provided by many languages, make software difficult to fit into one system prototype that is device independent. This is not to say that the software system cannot be made into a system diagram. The software system

diagram for DriveBot is shown in Figure 3-51. Bluetooth® commands are received by the Bluetooth 4.1, also known as Bluetooth Low Energy (BLE) stack, which is connected via a tunnel to the electrical system. These commands are passed to the control logic. Depending on the commands and the state of the robot, a Pulse Width Modulation (PWM) signal is created and sent out the left and right motor pins, which are connected to the electrical system.

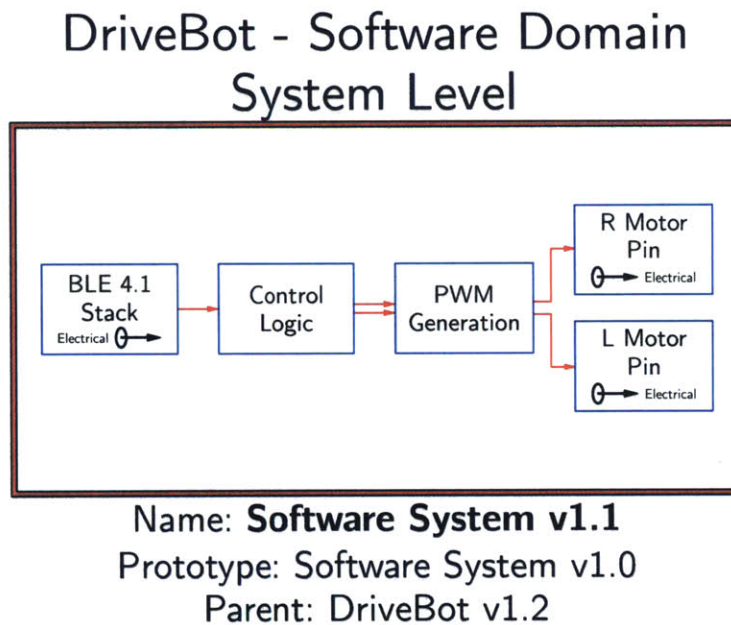


Figure 3-51: Diagram of the software system. The prototype for this block is empty by default.

3.5 Completed DriveBot Diagram

Combining the mechanical, electrical, and software systems results in the final system diagram for the robot, shown in Figure 3-52. The diagram contains all of the connections that result from the robot. While this exact robot may not be particularly useful outside of teaching System Diagramming and robotics, with every robot diagrammed, the library of blocks that has been designed grows and grows. Almost every subunit that was used for DriveBot will be used again in robots that provide additional and varied utility.

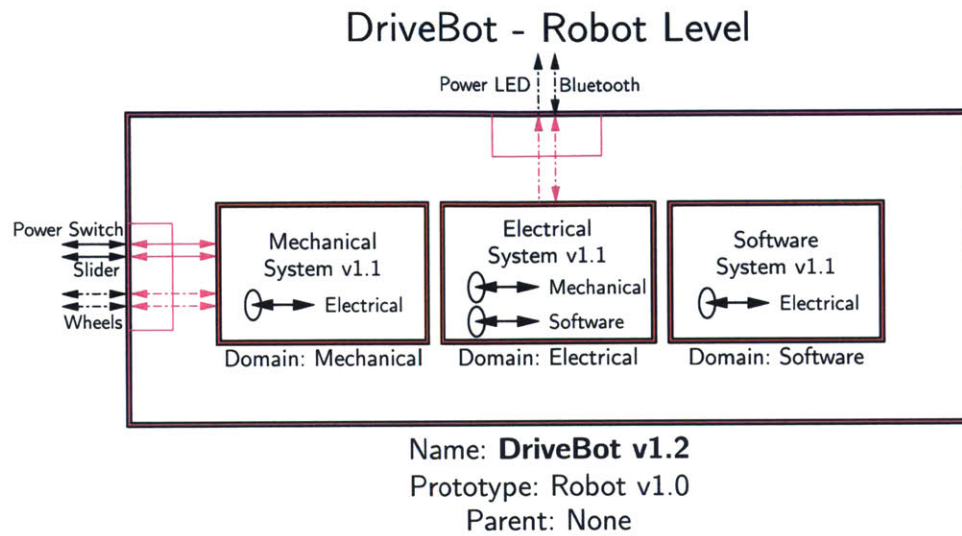


Figure 3-52: Final system diagram for DriveBot. Version 1.2 of DriveBot meets the definition of a robot, as it has mechanical, electrical, and software systems, as well as a controlled output.

Chapter 4

Using System Diagramming to Design MICA Blocks

The MICA (Measurement, Instrumentation, Control, and Analysis) Project is an ongoing research effort in MIT's BioInstrumentation Laboratory that aims to bring powerful, simple-to-use, wireless sensors and generators into the world of education [6]. As education gets augmented by software-based teaching, it is essential that hardware continues and expands into educational demonstrations and lessons. A core tenet of the MICA project is that students learn best through hands-on, measurement-driven education. Real-life experiments can provide a deeper intuition and a stronger motivation, than purely abstract lessons. By having a set of easy-to-use sensors and generators, individual students can quickly create an experiment, measure the results, and then analyze the results. When coupling this individualized hardware approach with software teaching methods, a new method for teaching students of all ages and backgrounds is possible, one which embodies the MIT motto, *mens et manus*, mind and hand.

4.1 System Diagramming

A sizable catalog of MICA sensors has been created by various researchers to date [7]. A major drawback of the previous version of MICA sensors is the difficulty of adapting

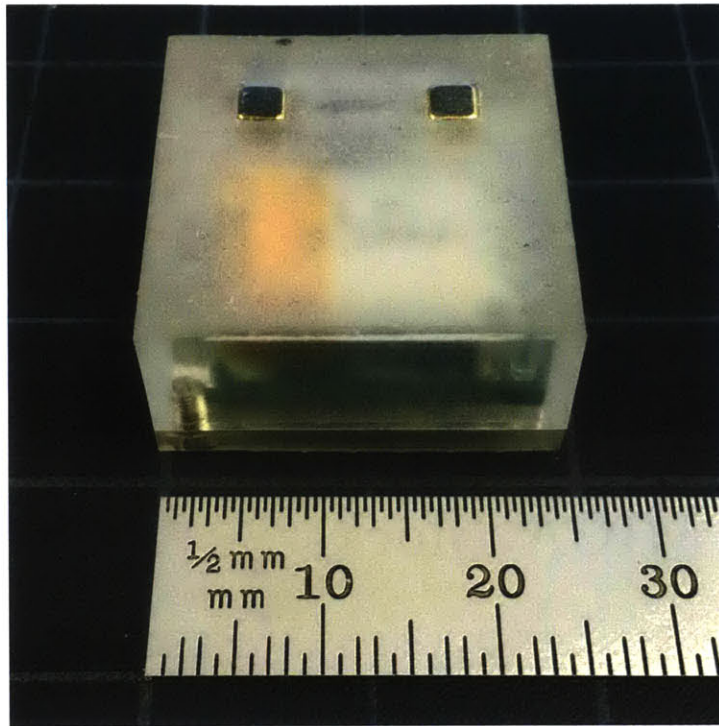


Figure 4-1: A MICA Inertial Measurement Unit (IMU) block. The two gold plated magnets can be seen at the top of the block, and the battery can be seen through the translucent enclosure.

the existing sensor designs to accommodate new designs. By applying the principles of System Diagramming that were established in the previous section, MICA blocks can be more systematically developed by utilizing the Electronics System prototype, and by populating the prototype with MICA subunits. These subunits can be recycled throughout multiple designs, resulting in an overall decrease in development time for creating new blocks. In addition to decreasing the time that it takes to incorporate new sensors, using the Electrical System prototype makes it easier to upgrade existing sections of a MICA block for new, higher performance blocks, and thus enhanced functionality.

Each physical MICA block represents one block of the Electrical System at a given level. For example, there are MICA *system* blocks, *module* blocks, and *unit* blocks, each of which performs differently based on the design intent and the level of operation. By chaining blocks, such as module blocks, together, functionality can be

expanded, resulting in a fully functioning electrical system. This concept is illustrated in Figure 4-2.

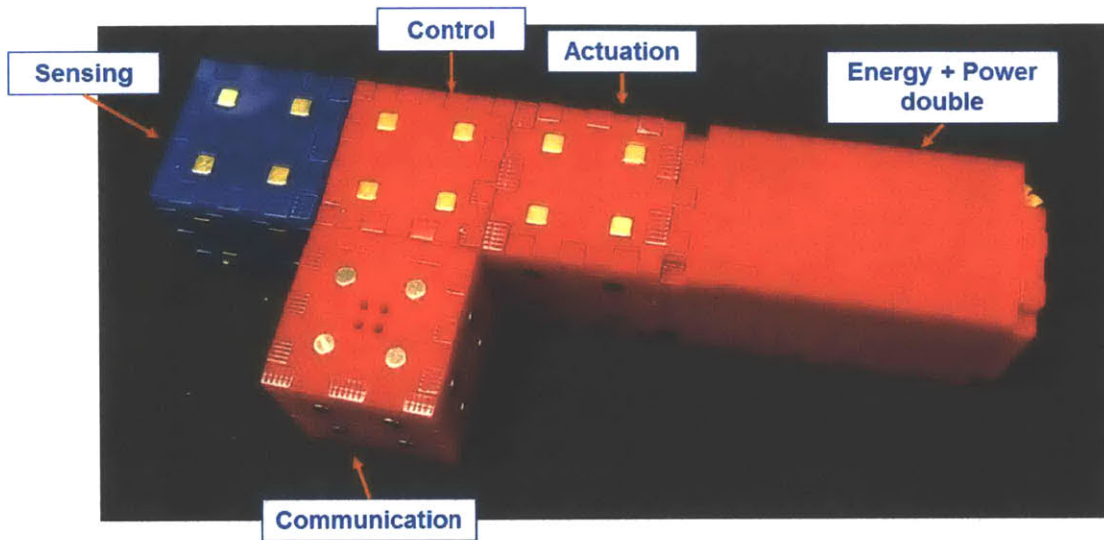


Figure 4-2: MICA blocks can be snapped together via electrically conductive magnets, to expand the functionality of the MICA system.

4.2 Electronics

MICA blocks can be seen as the electrical system building blocks for robotics. Blocks do not have to be used for strictly robotic applications; they can be used for a wide variety of sensing applications. Multiple revisions of a MICA Inertial Measurement Unit (IMU) Block have been developed throughout the process of merging System Diagramming with MICA. The final version of the IMU block can be seen in Figure 4-3. The IMU block's dimensions are $25\text{ mm} \times 25\text{ mm} \times 12.5\text{ mm}$. An IMU is a fusion of three different sensors, an accelerometer, a gyroscope and a magnetometer in this case, all integrated into one IC, for example, a BOSCH® BMX055 [8]. The cube connects wirelessly via BLE, which is achieved by the use of a Cypress Programmable System on Chip® (PSoC) 4 BLE module [9]. This MICA block transmits IMU data over the BLE connection to either a mobile device, or to a web application, where the data is stored for analysis and manipulation. On the back of the MICA IMU are two

gold plated rare-Earth magnets. These magnets serve as the mechanism for attaching the MICA IMU to external objects, as well as the electrical inputs for charging the battery.

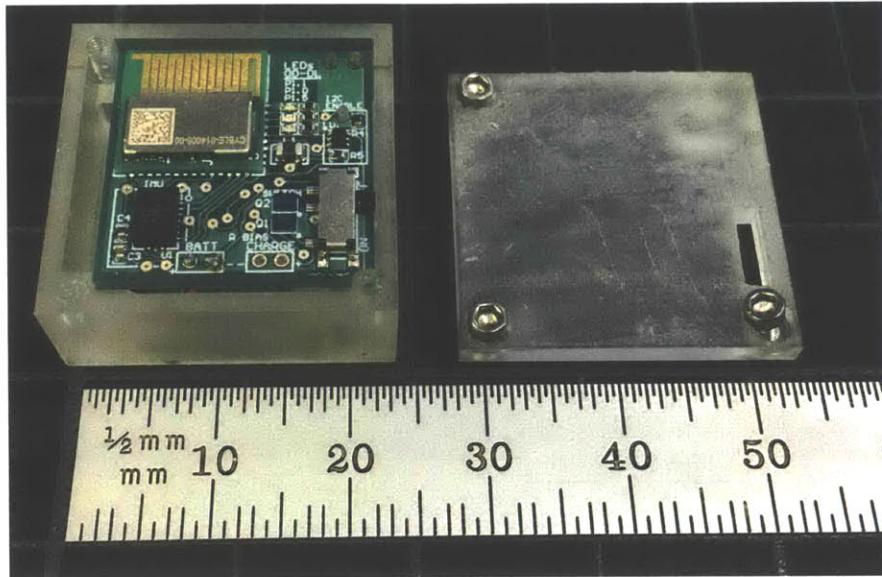


Figure 4-3: A MICA IMU with the top cover removed.

4.2.1 Schematic Design

One of the main benefits of using System Diagramming with electronics design, is that the electrical schematic and the System Diagram become one in the same. In Figure 4-4 the System Diagram of the IMU block can be seen. This System Diagram is the schematic that was used in Altium Designer[®] [11] to create the circuit board as seen in Figure 4-3. On the System Diagram of the IMU are the familiar blocks of an Electric Module prototype. Because the MICA IMU block is a Sensing Module, the Critical Block of the module is the Sensing Unit, which contains the IMU IC. One slight deviation from the module prototype is that the Communication and Control Blocks have been replaced by a Communication-Control group. This is because the critical control component is in fact the same component as the critical communication component, i.e. the PSoC 4 BLE. The PSoC contains a BLE subsystem as well as an ARM[®] Cortex M0 microcontroller [12]. The integration of the BLE and

microcontroller allows for a convenient package that saves space in the final implementation. Previous versions of the MICA IMU contained separate controller and communication units.

The inputs to the electrical system can clearly be seen from the System Diagram. There are two power inputs to the system for charging the Lithium-Ion battery that powers the MICA IMU. The IMU IC has a tunnel to the Mechanical Domain, which is where the forces to the system are presented. The signal from three differently colored LEDs are the only outputs from the system. However, the bulk of the information travels out of the system through the communication port via Bluetooth. Each of the units in the MICA IMU schematic can be expanded to the subunit level, and then the component level. The full System Diagram for the MICA IMU is shown in Appendix A. A recycling symbol on a block indicates that the block is part of a library of block that can be reused multiple times.

4.2.2 Board Design

Once the electrical System Diagram for the MICA IMU circuit is fully defined, the board is ready to be laid out. Laying out the board is the same process as defining the mechanical System Diagram for the circuit board. The board is laid out with components grouped by subunit. Once a subunit has been laid out, a snapshot of the layout is saved and associated with the subunit. Whenever another copy of that subunit is used again, the only work that has to be done on the circuit is connecting the inputs and outputs. In Altium, this is achieved by using the “rooms” functionality. Once all of the subunits are individually laid out, they are placed on the circuit board and connected together. Each electrical component is represented in two ways: an electrical schematic to denote functionality and a mechanical CAD model, to represent the physical footprint. The resulting CAD model can be seen in Figure 4-5. The subunits on the board can be seen in the white silk screen on the board in Figure 4-6.

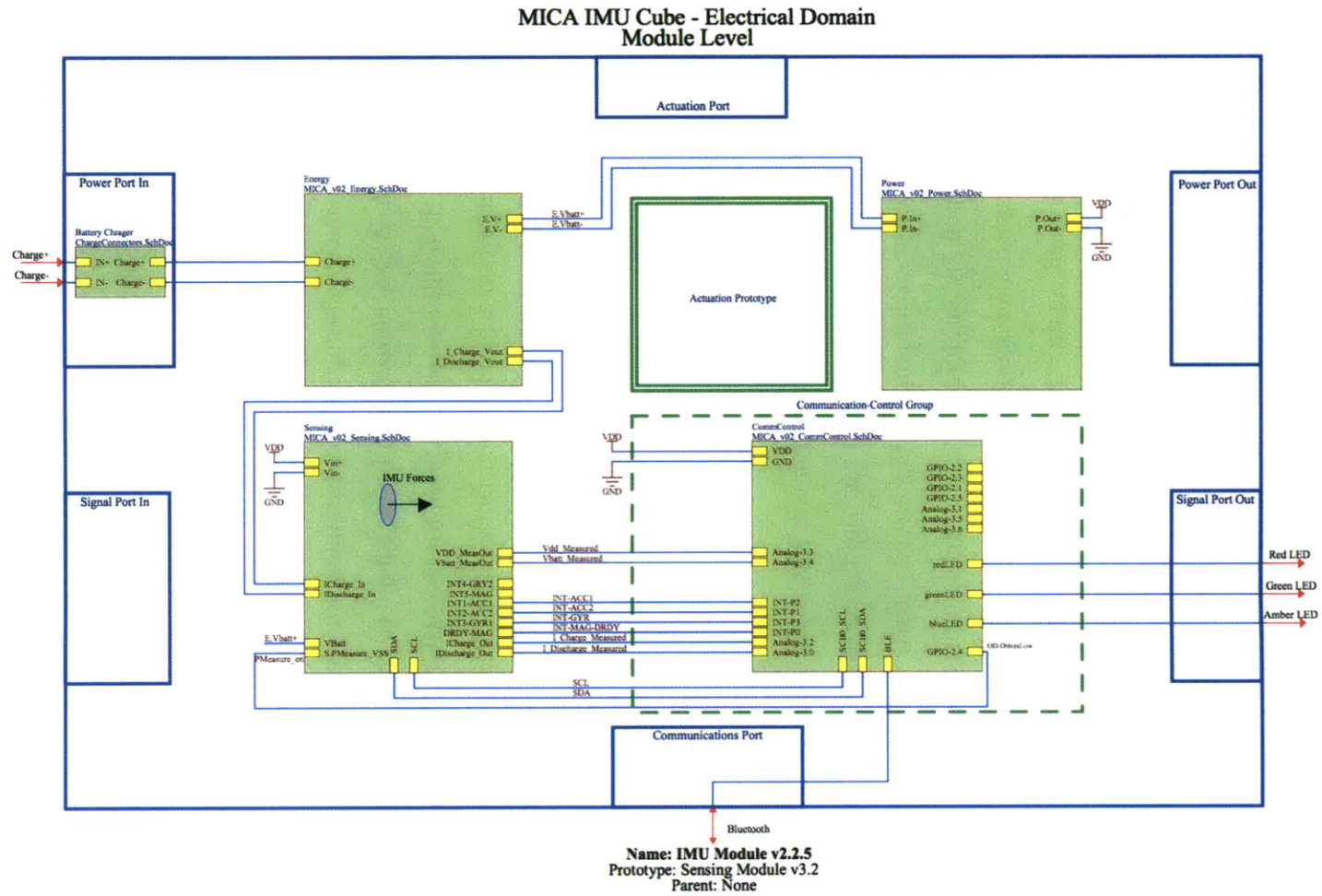


Figure 4-4: The top level circuit schematic in Altium, which is the same as the top level electrical System Diagram for the MICA IMU.

made on a laser cutter, using the geometry generated by the mechanical models of components, and their position generated by Altium. After the stencils are cleaned, an aerosol adhesive is sprayed on the back. Then, with the use of a microscope, the stencil is placed on the PCB (Figure 4-7) and solder paste is applied to the stencil using a razor blade. At this point the stencil is removed, leaving solder paste on the component pads. The components are then placed onto the board by hand with tweezers. Finally, the board is placed into a reflow oven to solidify the solder paste. After the first side is populated, the same steps are repeated on the opposite side of the board. An adhesive does not need to be placed on the first side when the second side is being reflowed, as the surface tension from the solder paste prevents the components from falling off, even as the solder paste becomes liquid again.

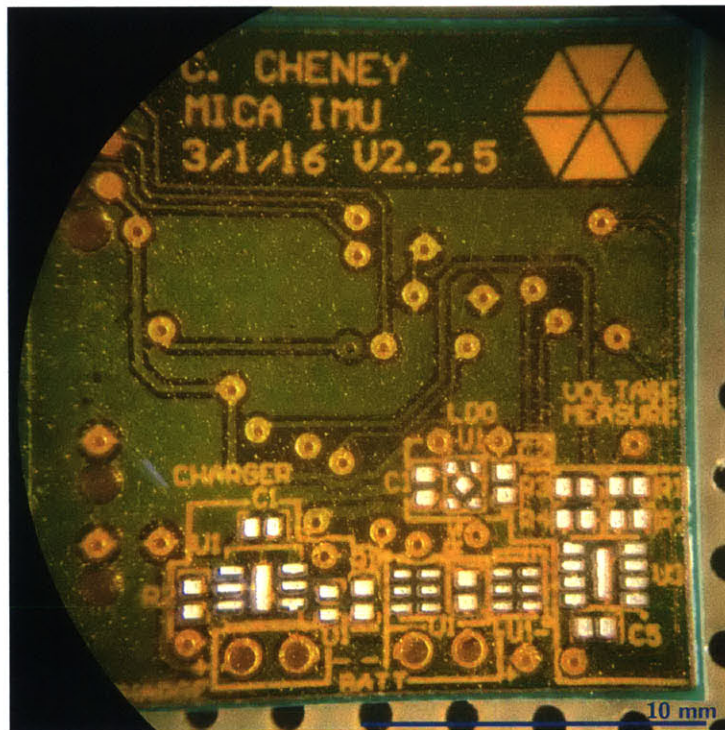


Figure 4-7: An orange Kapton stencil is adhered to the unpopulated MICA IMU circuit board, seen through a microscope. The stencil is laser cut based on the mechanical component models, and their placement on the PCB.

4.3 Mechanics

Having an accurate PCB model is crucial for designing an enclosure for each MICA block. Each enclosure must have a mechanism for closing the block, a way of fixing the PCB in place, a mechanism for fixing the block to the surrounding system, a place for the battery, and a port for charging the battery. The resulting CAD model for the MICA IMU can be seen in Figures 4-8.

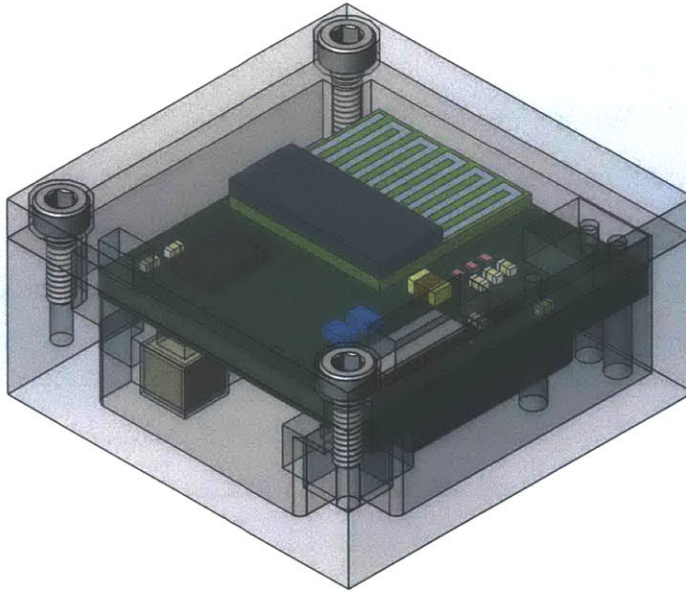


Figure 4-8: A CAD model of the final MICA IMU. The enclosure is 3D printed using stereolithography.

4.3.1 Form Factor and User Experience

One of the major considerations of the MICA project is the user experience for students. A major goal of MICA is to completely remove burdensome wires from experiments. To achieve this, there must be a mechanism for charging the MICA block that does not use a plug. For the MICA IMU, there are two gold plated rare-earth magnets on the back side of the block, as seen in Figure 4-9. These magnets allow the IMU to be mechanically attached to a target, but also serve as the electrical connectors for

the battery charging system. The IMU is attached to other gold plated magnets that are connected to an external battery, or power supply. The magnets hold the block in place, while current flows through the gold. Charging currents in excesses of two amps have been sustained, without any adverse effects on the permanent magnets.

Another user experience area is turning the block on and off. While a physical switch is built into the MICA IMU for development purposes, the goal for powering on and off is to use a feature recognition functionality of the accelerometer. The BOSCH BMX055 was selected in part because of its $2.1 \mu\text{A}$ total supply current in suspend mode. The accelerometer is always kept on and in suspend model, even when the MICA block is “off.” When a user double taps the block, the accelerometer hardware interrupt is triggered, causing another interrupt to trigger in the system controller, which proceeds to wake up the system from a low power mode. When users want to turn off the MICA IMU, they do so using the mobile or web application, as discussed in the next section.

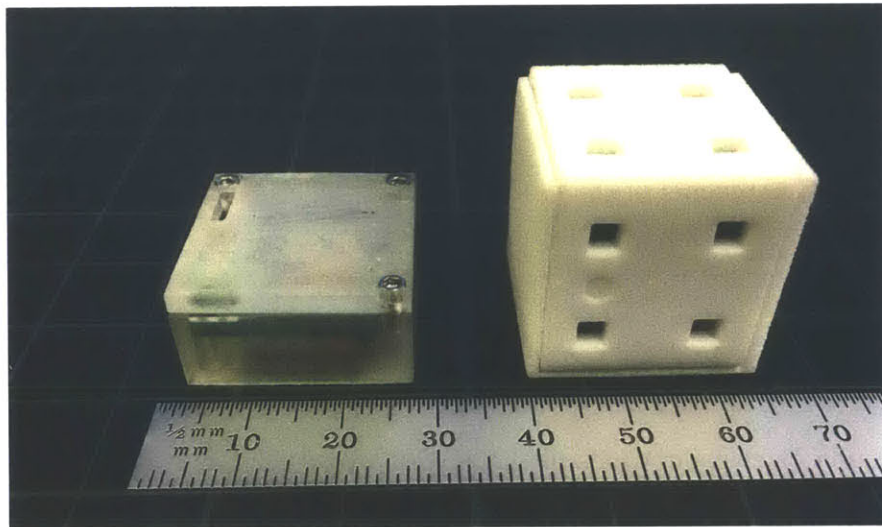


Figure 4-9: (Left) The final MICA IMU and (right) a previous version. Both blocks have gold plated permanent magnets for mechanical mounting and battery charging.

4.4 Software

For MICA blocks to be useful in teaching environments, they need to be accessible and simple to use. Previous versions of MICA blocks [7] had a built-in Organic LED (OLED) screen that acted as a user interface for the cube. However, the ubiquity of mobile devices has resulted in the on-board screen in MICA blocks becoming redundant. Eliminating the screen from MICA blocks saves on space, power consumption, and lowers development overhead complexity related to controlling the screen. A mobile Android application and web application are used now to control the device wirelessly, via BLE.

4.4.1 Embedded Software

The code that is incorporated by the System Diagram for the MICA block is the embedded software. This includes the code that controls the IMU, manages power and BLE communications, and measures power consumption. For the MICA IMU, the embedded software is written using Cypress' PSoC[®] Creator[™] [10] integrated design environment (IDE). Creator is broken up into three main parts, each of which lends themselves to System Diagramming. First, a user drags and drops virtual components into the Creator schematic (Figure 4-10), which is similar to using blocks from a System Diagramming library. These components are then wired to the specific pins that were used in the electrical system, through Creator's Design Wide Resources tab. Finally, components are controlled using application programming interface (API) calls in C code.

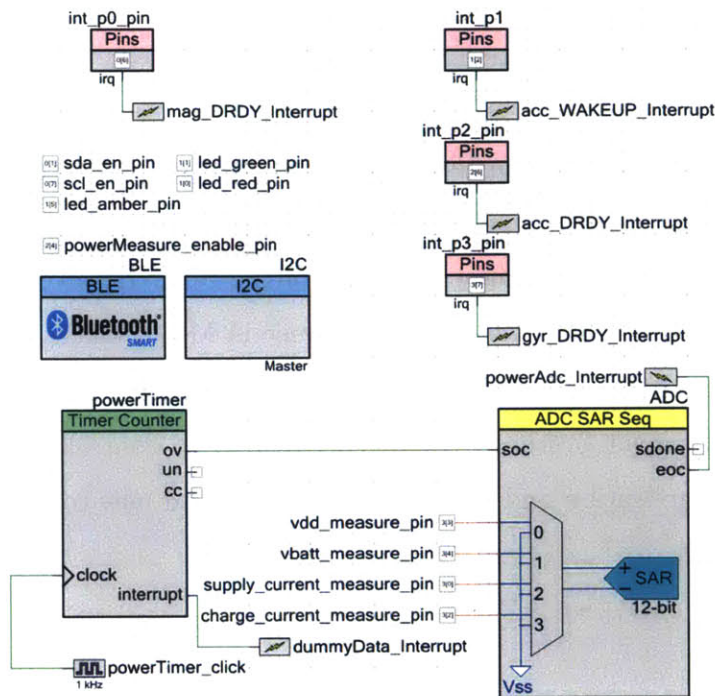


Figure 4-10: A screenshot of the schematic in PSoC Creator that controls the MICA IMU. The schematic shown closely relates to the software system diagram of the MICA IMU.

4.4.2 MICA Mobile

One of the major advantages, in addition to decreased power consumption, of using BLE over another wireless protocol, is that a receiving device can communicate with a MICA block without having a special wireless adapter. It is estimated that by 2018, there will be 10 billion BLE enabled mobile devices [13]. Accordingly, a mobile application (MICA Mobile) has been written for the Android operating system. This application connects to MICA blocks via the mobile device's BLE, where a user can control settings on the device and start data logging from a given sensor. The data received from the block are graphed in real-time on the screen and simultaneously sent to a server in the BioInstrumentation Lab (Figure 4-11). Once the data are on the server, it is synced to the cloud and can be retrieved by others for inspection and analysis. Getting the data into the cloud is a crucial step for education, as it allows

students to easily compare and collaborate on data sets. Figure 4-12 shows a previous version of the MICA IMU connected to MICA Mobile as the block is moved.

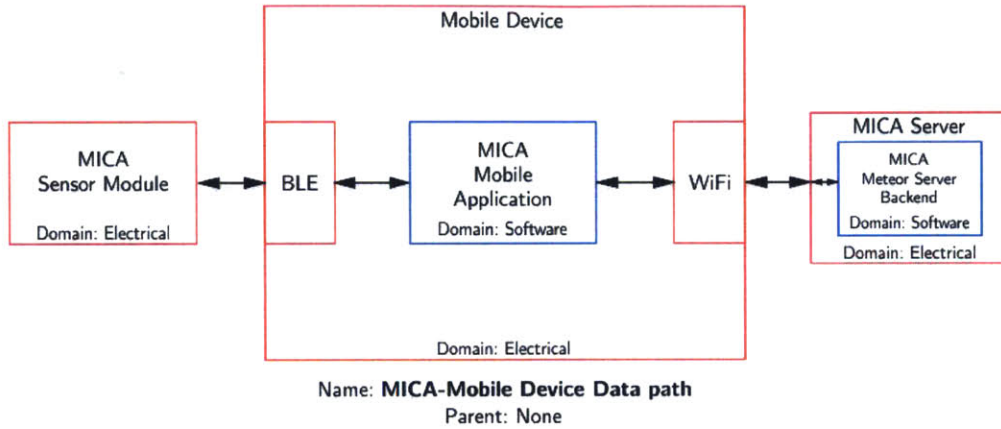


Figure 4-11: The software System Diagram for the sensor data when the MICA block is connected to MICA Mobile.

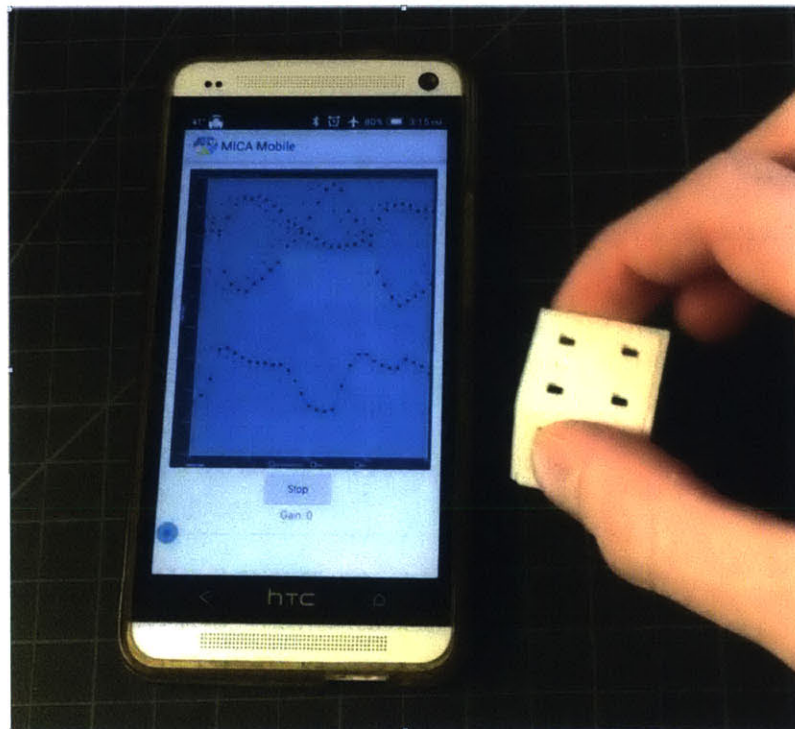


Figure 4-12: Acceleration data are plotted in MICA Mobile as a previous version of a MICA IMU is shaken.

4.4.3 Web Application

While mobile devices that are BLE enabled are the wave of the future, steps must be taken to ensure that non-BLE enabled devices, such as some laptops and desktop computers, can be used for MICA purposes. To make these devices backwards compatible, a BLE to USB dongle is plugged into the computer. This dongle then connects with a custom Google Chrome [14] application, *MICA Chrome*. MICA Chrome is used to control the block in a similar manner as MICA Mobile. MICA Chrome posts the data received from the sensor to the MICA server. Users then log onto a MICA web application, written using the Meteor framework [15], to interact with the datasets (Figure 4-13). A photo of the MICA IMU being moved while data is plotted in real time on the MICA web application is shown in Figure 4-14.

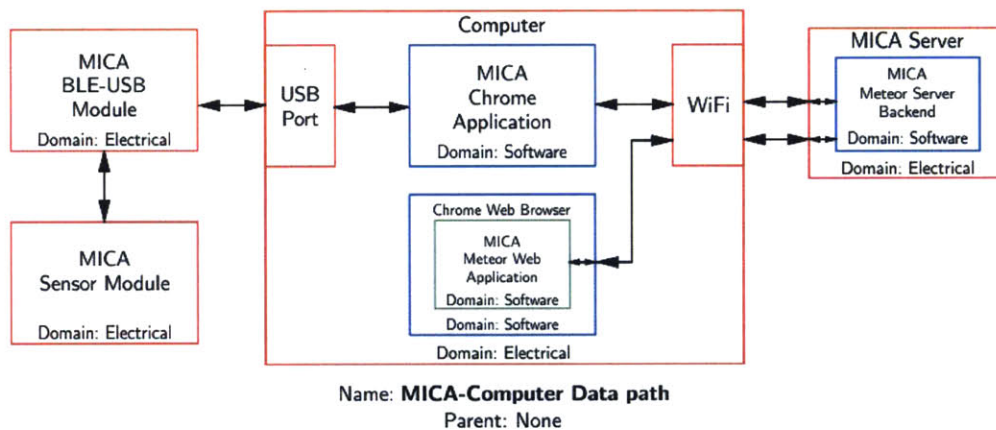


Figure 4-13: The software System Diagram for the sensor data when the MICA block is connected to a BLE-USB dongle and MICA Chrome.

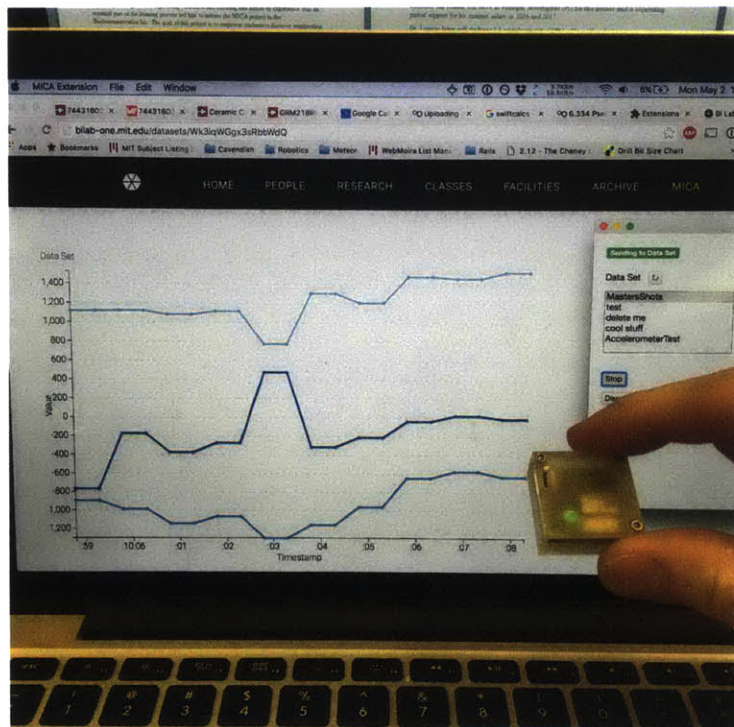


Figure 4-14: As the MICA IMU is moved acceleration data is plotted in real time on the MICA web application.

Chapter 5

Conclusion

Block diagramming is used to design systems across many fields, with varying degrees of rigor. In this thesis, a set of self-consistent rules was developed that enables System Diagramming to be standardized regardless of the field they are describing. System Diagramming allows for the creation of *prototypes*, which provide a framework for System Diagramming blocks to be placed in, further increasing the standardization of a system. A System Diagram prototype specific to robotics was presented with several examples.

System Diagramming and the robotic prototype were merged into the existing MICA project, which proposes to bring a suite of wireless sensors and generators to the world of education. Using System Diagramming with MICA enables MICA blocks to be developed and revised rapidly and also clearly defines a blocks role in a larger system.

A set of MICA blocks were developed that used Bluetooth 4.1 which will allow for a more widespread adoption of MICA. A mobile application was written for the Android operating system, as well as a web application for controlling and interacting with MICA cubes. These applications allow for MICA to be a cloud based project, which is well suited for the education environment.

System Diagramming and MICA have come a long way, but both still have tremendous areas for growth. The creation of System Diagrams is quite time intensive and tedious. A web application should be written to help automate the process. This

would allow for team based design of systems, and would allow for the full potential of diagram versions to be used. Other cloud based softwares should be integrated into the System Diagramming application, to the point where the System Diagram is the link between the electrical CAD, mechanical CAD, and software IDE.

In terms of MICA, the web application and Android application both need significant development efforts to become mature. An iOS application needs to be implemented as well. Furthermore, a large catalog of MICA sensor and generator cubes needs to be developed. In order to achieve this, effort needs to be put into creating MICA tools that are helpful in the development of other MICA blocks.

Bibliography

- [1] Linear Technology, Standalone 750mA Li-Ion Battery Charger, LTC4065 datasheet, <http://cds.linear.com/docs/en/datasheet/406544f.pdf>, 2016
- [2] Boston Dynamics, Atlas - The Agile Anthropomorphic Robot. http://www.bostondynamics.com/robot_Atlas.html, 2016
- [3] Neville Hogan. 2.141 Modeling and Simulation of Dynamic Systems, Fall 2006. (Massachusetts Institute of Technology: MIT OpenCourseWare), <http://ocw.mit.edu> (Accessed 6 May, 2016). License: Creative Commons BY-NC-SA
- [4] Tonti E., Why starting from differential equations for computational physics?, *Journal of Computational Physics*, 257(2014) 1260-1290.
- [5] Tonti E., A direct discrete formulation of field laws: The cell method. *CMES - Computer Modeling in Engineering & Sciences*, 2(2):237-258, 2001.
- [6] Spanbauer, A.; Wahab, A.; Hemond, B.; Hunter, I.; Jones, L. *Measurement, instrumentation, control and analysis (MICA): A modular system of wireless sensors*. IEEE International Conference on Body Sensor Networks, 17-21, 2013.
- [7] Spanbauer, A; *MICA Workspace: A Symbolic Computational Environment for Signal Analysis*. Thesis. MIT, 2013
- [8] Bosch Sensortec, Small, versatile 9-axis sensor module, BMX055 datasheet, 2014
- [9] Cypress Semiconductor, EZ-BLE™ PSoC® Module, CYBLE-0140080-00 datasheet, 2016

- [10] Cypress Semiconductor, PSoC[®] Creator[™] Integrated Design Environment (IDE), <http://www.cypress.com/products/psoc-creator-integrated-design-environment-ide>, 2016
- [11] Altium Designer. <http://www.altium.com/>, 2016.
- [12] ARM Cortex. <http://www.arm.com/products/processors/cortex-m/cortex-m0.php>, 2016.
- [13] ABI Research. “Installed Base of Bluetooth-enabled Devices Worldwide in 2012 and 2018 (in Billions).” Statista - The Statistics Portal. Statista. August 2013. Web. 6 May 2016. <http://www.statista.com/statistics/283638/installed-base-forecast-bluetooth-enabled-devices-2012-2018/>
- [14] Chrome. <https://www.google.com/chrome/>, 2016.
- [15] Meteor. <https://www.meteor.com/>, 2016.