B L A C K W O O D D E S I G N S

Requirements

SPECIFICATION & ANALYSIS

AUTHOR RANDALL MAAS

OVERVIEW This is fascicle explores my ideas how to write and analyze a specification, that lends to a direct realization in hardware and software.

BENEFITS Straight forward, testable designs, predictable behaviour

USES
Embedded systems
Control systems

FILE: G:\My Documents\Requirements\Specification & Requirements analysis - 2018-5-30.doc

RANDALL MAAS has spent decades in Washington and Minnesota. He consults in embedded systems development, especially medical devices. Before that he did a lot of other things… like everyone else in the software industry. He is also interested in geophysical models, formal semantics, model theory and compilers.

You can contact him at randym@randym.name.

LinkedIn: http://www.linkedin.com/pub/randall-maas/9/838/8b1

Preface

This fascicle – short document – focuses on developing a style of specification that that lends itself to a direct (or nearly so) realization of in hardware and software.

This is a fascicle covering a specification, its definitions, requirements and the analysis of these. It does not cover:

- Project management
- Requirements process or management
- Design architecture
- Engineering development process

1 ORGANIZATION OF THIS FASICLE

- CHAPTER 1: PREFACE. This chapter describes the other chapters and further reading
- CHAPTER 2: SPECIFICATION OUTLINE. Provides an outline for the structure and contents of a specification.
- CHAPTER 3: REQUIREMENTS ANALYSIS. Describes how to analyze a specification and it's requirements.
- CHAPTER 4: REQUIREMENTS CHECKLISTS. This chapter provides checklists for reviewing requirements.
- CHAPTER 4: ELECTRONICS DESIGN DESCRIPTION OUTLINE. Provides an outline for the structure and contents of a design description of the electronics.
- CHAPTER 5: SOFTWARE REALIZATION. This chapter describes how to convert the requirements into a (partial) implementation.

APPENDICES. The part provides supplemental material

- APPENDIX A: ABBREVIATIONS, ACRONYMS, & GLOSSARY. This appendix provides a gloss of terms, abbreviations, and acronyms.
- APPENDIX B: COMMON NOUNS. This appendix lists common nouns, including peripherals and sensors.
- APPENDIX C: COMMON PROPERTIES. This appendix lists scientific units/dimensions.
- APPENDIX D: COMMON CATEGORICALS. This appendix lists the common categoricals.
- APPENDIX E: COMMON COMPARISON. This appendix lists the common comparables, comparatives, superlatives.
- APPENDIX F: COMMON, WELL-KNOWN EVENTS. This appendix lists the common well-known events.

- APPENDIX G: CODE COMPLETE SPECIFICATION REVIEW CHECKLISTS. This appendix reproduces checklists from *Code Complete, 2nd Ed* that are relevant to specification reviews.

2 REFERENCE DOCUMENTATION AND RESOURCES

Arora, Chetan; Mehrdad Sabetzadeh, Lionel Briand, Frank Zimmer. "Automated Checking of Conformance to Requirements Templates Using Natural Language Processing" *IEEE Trans Software Engineering*, vol 41, n10, 2015 p944-967

Gilb, Tom. *Planguage* 1988

Gilb, Tom. *Competitive Engineering*, 2005 Butterworth-Heinemann

NASA "Appendix C: How to write a good requirement" in *NASA Systems Engineering Handbook*

2.1 EARS: EASY APPROACH TO ENGINEERING

- Mavin, Alistair. *EARS Tutorial Easy Approach to Requirements Syntax*
- Rupp, Chris. "*Easy Approach to Requirements Engineering*" (EARS)
- Terzakis, John. *EARS: The Easy Approach to Requirements Syntax*

2.2 INSTRUMENTATION & SIGNAL PROCESSING

- Garrett, Patrick H. *Advanced Instrumentation and Computer I/O Design: Real-Time System Computer Interface Engineering,* IEEE Press, 1994
- Redmon, Nigel *Biquad Formulas* 2011-1-2 http://www.earlevel.com/main/2011/01/02/biquad-formulas/
- Smith, Steven W *"The Scientists and Engineer's Guide to Digital Signal Processing,"* Newnes, 1997, http://www.dspguide

3 THE DIFFERENT TYPES OF SPECIFICATION DOCUMENTS

The documents – or portions of documents – discussed here include:

A *high-level document* is a finite set of requirements documents, e.g. system specification, customer inputs, marketing inputs, etc. — *high-level document*

A *requirements document* is a set of requirements, and clear text explaining or justifying the requirements. A justification may base the requirement in other documents, such as research, standards, regulations or other laws. — *requirements document*

A *requirement* defines what an item must do, and is presented as text of a special form (to be discussed throughout this fascicle). — *requirement*

A *customer requirement* is a requirement in any of the top-level documents. — *customer requirement*

A *comment* is a text. The comment's text is not analyzed in this document — *comment*

A *test specification* is a requirements document that describes a set of tests intended to check that the product meets it requirements. This document defines: — *test specification*

- A set of *test requirements* that define what tests a product must pass. — *test requirements*

- A mapping, *tests*, that maps a test requirement to a set of requirements that it tests.

tests

A *design document* explains the design of a product, with a justification how it addresses safety and other concerns. TBD/Future needs a better description that better can link to input requirements.

design document

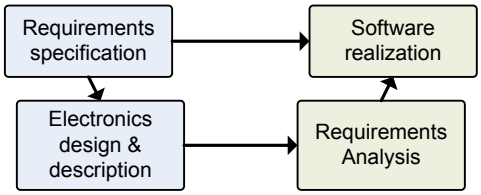

Figure 1: *Intended flow of documents*

Note: the analysis is only that the specification can be built and tested – that there is enough information to do so, and that it does not conflict. Analysis for whether the specification is correct is separate and not covered here.

A *test report* is a set of outcomes: <test id, product id, result> describing how a product performed under test. (The performance may vary with versions of the product)

test report

An *identifier* can refer to product, specific version of the product, a document, requirement, test, external document, or comment. In practice this is so important that each item is given a label.

identifier

A *trace matrix* defines two functions, forming a directed acyclic graph:

trace matrix

- Maps a requirement to the set of requirements that it directly descends from
- Maps a requirement to a set of requirements that directly or indirectly descends from it.

A *hazard analysis* is a set of documents that define:

hazard analysis

- Maps a harm to severity
- Maps a hazard x harm to likelihood

CHAPTER 2

Specification outline

This chapter describes the preferred outline for a specification:

- Main outline
- Synopsis & Front matter
- Overview
- Definition

1. MAIN OUTLINE

This chapter describes the preferred approach to writing a specification. The following is the outline for a specification:

1. Synopsis
2. Other front matter. These, if part of a larger document, should be placed in the preface or appendices of the larger work:
    a. Glossary, acronyms
    b. Related documents (documents that are part of the product)
    c. References, resources, suggested reading
3. Overview
    a. Diagram, calling out key items referred to
    b. Walk-thru of the intended use of the product
    c. Potted product specification
    d. Feature list
4. Definitions
    a. Kinds and entities
    b. Properties and property values
    c. Comparables, Comparisons, and Comparatives
    d. Actions
5. Goals, Objectives
6. Requirements
    a. Functional requirements
    b. Detailed logical controller specification
    c. Performance requirement
7. Analysis to support the design or requirements

[alternate forms – all of the definitions up front / or segmented out per group]

1.1. EDITING

Tip: use MS Word's grammar checker, and readability statistics.

2. SYNOPSIS AND FRONT MATTER

THE SYNOPSIS. A one or two paragraph synopsis

THE RELATED DOCUMENTS, SPECIFICATIONS. Include a designator for each document. Use this through the remainder of this specification to refer to the document.

THE REFERENCES, RESOURCES, SUGGESTED READING. Include a designator for each document. Use this through the remainder of this specification to refer to the document.

THE ACRONYM AND GLOSSARY TABLES. Define all acronyms, terms and phrases. We have all seen documents that include definitions for simple, common items (such as a LED), while not defining specialized items (such as "adaptive linear filter") referred to heavily in a document. Don't do this.

Tip: The acronyms and glossary are well suited for reuse in many projects. Make a stock document with the most common terms and potted definitions to be included in each project.

3. OVERVIEW

The document should include an overview of the product. In practice, this should be two (interrelated) overviews. The first is a big picture description – from the perspective of the final product that the customer or operator receive. The second description is specific to the sub-assembly product. In some cases, additional overviews would be a good idea to bring into focus a very specific element within the big picture.

THE OVERVIEW should include:

- A diagram, calling out the particular items that will be discussed
- A description of what the product is, what role it serves, with some hint of its benefit
- A potted specification of the product

THE DIAGRAM. Each item in the document rest of the document should be called out thick call out lines, often red. Number the items, and describe each in the text. Include the name or designator that will be used throughout the rest of the document.

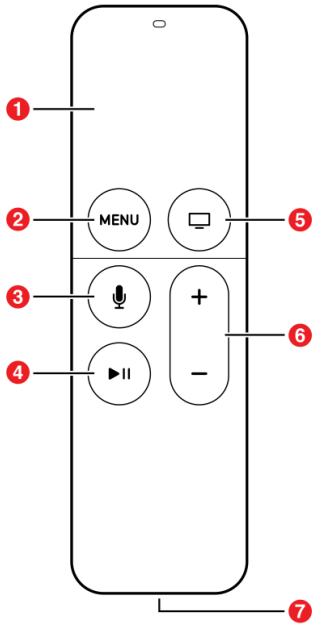

Figure 2: *Example diagram with call-outs*

The elements external to the remote control are:

	Element	Description
1	Touch surface	Swipe to navigate. Press to select. Press and hold for contextual menus
2	Menu	Press to return to the previous menu.
3	Siri/search	Press and hold to talk. Opens the onscreen search app.
4	Play/pause	Play and pause the media.
5	Play/pause	Press to return to the home screen. Press twice to view open apps. Press and hold to sleep.
6	Volume	Controls the TV value
7	lightning connector	Plug-in for charging

Table 1: *Example table of diagram elements*

THE DESCRIPTION of the product should capture what the product is, and its role. The descriptions should include a walkthrough of its primary operations. This overview should strive to be clear, and easy to understand. Assume that until the reader has seen this description the use is non-intuitive. Select the language carefully, and define each of the terms used.

THE FEATURE LIST(S). Include a list of the key features. For example:

- Can control the TV volume
- Can control media playback with a “Jog” control
- Can search for media with a voice command

Tip: Similar (end) product-level diagrams and description may be used in many projects. Make a template document with a typical diagram and description. This can be used as a starting point in each related project.

4. DEFINITIONS

The next major section should define the terms and key phrases:

- Kinds and instances
- Properties and property values
- Comparisons, Comparables, Comparatives
- Classification table
- Constraints on values
- Actions
- Events

4.1. KINDS AND INSTANCES

Define the pieces, etc. that will be specified later. Be clear whether a noun phrase is…

- a specific, singular instance in the product; or
- A kind or class of thing that there may be many instances of.

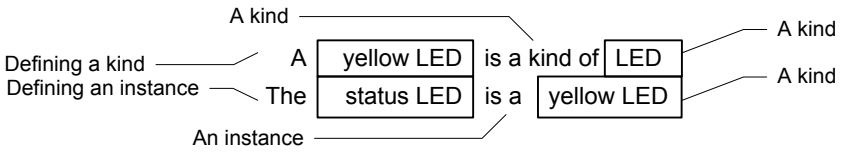

Figure 3: *Kinds and instances*

The *kinds* – and perhaps several of the instances – should be included in the glossary. These are often common nouns. See Appendix B for well-known common nouns that need not be defined each time.

The *instances* (and their kinds) may be well suited for being listed in a table. These are often proper nouns.

The designator used to declare a kind or an instance should be used consistently throughout the document. Avoid employing only a portion of the designator, pronouns or indexicals to refer to it.

4.2. PROPERTY DEFINITION AND POSSIBLE VALUES

If there is a need to define properties, include a (sub)section to do so. (There are many trivial, well-known properties; see Appendix C for examples. Usually, one does *need* not to define these again. If there is any doubt, however, define the property.)

Define property by its name and its set of possible values. The set of values that a property can take on are usually one of two kinds:

- Categorical (aka nominal),
- Scalar value; these often have a dimension and unit.

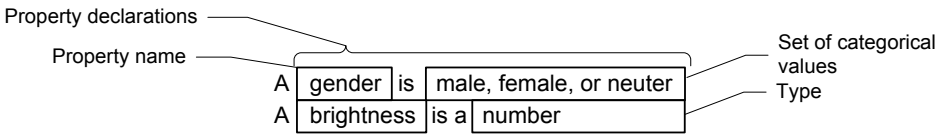

Figure 4: *Property declarations*

The properties should be included in the glossary. List the properties that a kind or instance has. Not all properties have names.

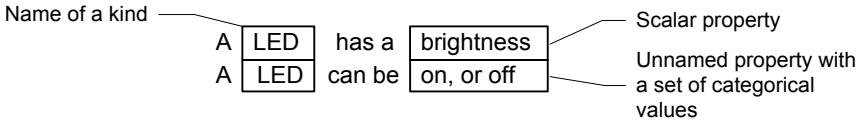

Figure 5: *Properties of kinds and instances*

The properties that a kind or instance may be well suited for being listed in a table.

4.3. PROPERTY VALUES

Define the property values for each given thing, if they have not been defined yet.

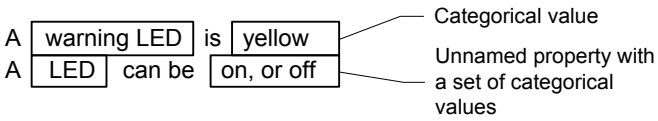

Figure 6: *Property declarations*

The property values are often well suited for being listed in a table:

property	min	max
image width	240 pixels	240 pixels
image height	216 pixels	216 pixels
data rate	9600bps	115000bps
stop bits	1	1
operating current	-	130mA
operating voltage	3.3v	6v

Table 2: *Example table of property value ranges*

Note that the property's meaning is defined earlier. The definition includes whether the subject provides the given property value, or operates correctly values with values in the range.

4.4. DEGREES OF COMPARISON: COMPARABLES, COMPARATIVES AND SUPERLATIVES

comparisons and superlatives

In some cases it is useful to define comparable adjectives on a kind or instance. (This is relatively rare, as it is unnecessary most 'well-known' comparisons on scalar properties. See Appendix E for well-known comparisons.) These relate a property's value to a binary relation.

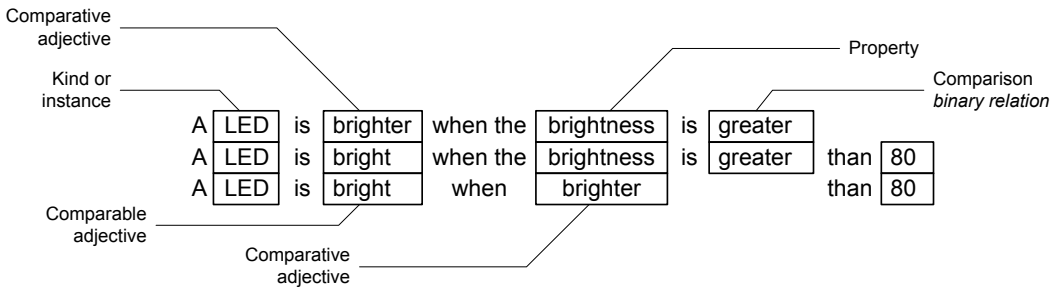

Figure 7: *Comparative definitions*

Superlatives. such as "*brightest* of a set." That element is compared against all others, and found to have passed the tests

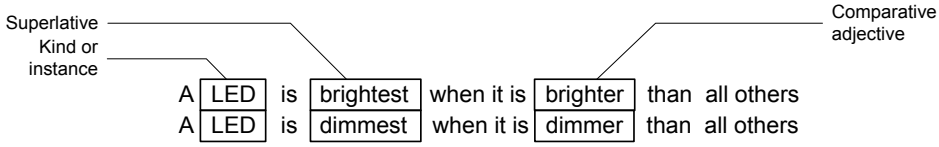

Figure 8: *Superlative definitions*

4.5. DERIVED PROPERTIES: MAPPING TO STATES OF PROPERTIES TO AN ENUMERATED VALUE

classification table

This section how to map states of properties to an enumerated (aka nominal or categorical) value. These are often used to identify a state, classify, or action (from a set). Decision tables (aka classification table) are commonly used to maps the input conditions to an output state:

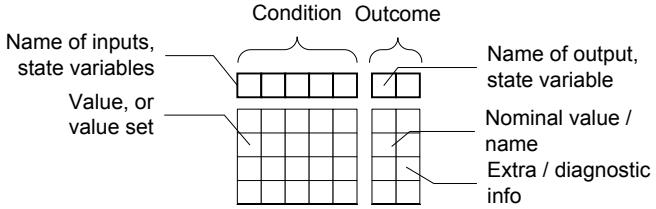

Figure 9: *Property lookup table*

The rows should not compete or conflict.

- The *columns names* identify the instance and property. A column name may also specify that it refers to the value at the previous time step. (Without this specification, the column refers to the value at the current time step.)
- Values (or sets of values), are in the cells. Note: expressions such as "< 23" are treated as a set of values (the set all numbers less than 23)
- Values not defined in the conditions are meant to be interpreted as "any value is acceptable."
- Cell may also specify another property name or value. Note interpretation order in this case.
- All outcomes must have a defined value.

Describe the default value to use when no row matches

4.6. CONSTRAINTS ON VALUES

constraints

Example of a constraint:

> The sum of the current must be less 20mA.

4.7. ACTIONS

actions

The key actions & activities of the product should be defined. Fortunately, not all actions need be spelled out in detail. Examples of actions whose detailed steps may be skipped:

- If the action is well-defined in a sourced document,
- If the action is trivial, obvious and states the outcomes, such as the light being on.

The elements of a good action description:

- Actions should have their outcome stated, or be clear from the text. For example "turn the yellow LED on" is clear that the resulting state of the yellow LED is "on."
- If the action is defined in a sourced document, reference that document and its relevant portions explicitly.

The state of the world when it ends.

Actions can take time, and so have a beginning, middle, and end. Specify the minimum and maximum duration. Specify testable elements or parameter values during the time from beginning to end. These requirements are usually definitions of what vague qualities such "smooth" mean. For example as ramping.

[diagram]

Figure 10: *Example diagram of activity qualities durations*

The maximum and minimum slope.

Actions are often built on other requirements (and actions), along with parametric requirements to accomplish the action correctly.

Describe the steps to take to carry out action. Each step will [?] to be tested. If description is based on other actions be sure to define them too.

4.8. EVENTS

[todo describe named events]

5. REQUIREMENTS

requirement

A *requirement* defines what an item must do, and presented as text in a special form. Requirements are the means of assuring correct operation in the process of design and construction. The purpose of a requirements-driven process is to promote a cultural discipline that focuses and attention to build the quality products. This quality is often focus on safety aspects.[1]

[1] Although it would seem to be a process focused on building the product the parent organization wants, the process was only (historically) shaped to meet safety goals. There were (and are) other processes, but they did less well with safety.

A text should be separated from the text like:

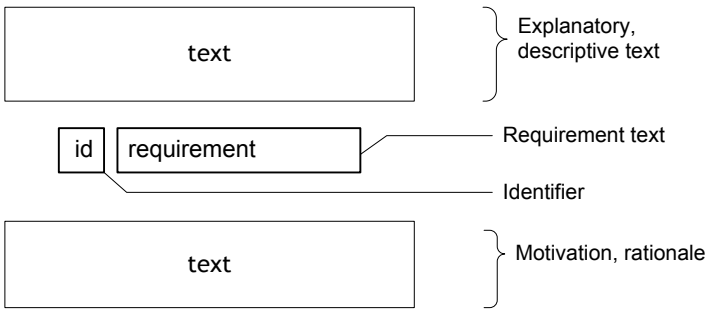

Figure 11: *Typical requirement text*

The presentation of a requirement in the text should include:

- *Clear demarcation* of the requirement. For instance, place the requirement on an indented line, by itself.
- A means to *uniquely identify* or refer to the requirement. It is important to be able to identify the requirement be discussed. The requirement will be referred to in other documents, trouble tracking, etc.
- A brief *summary* of the requirement and its purpose or intent.
- *The actor* who carries out or meets the requirement. The actors should be defined earlier in the section or the document.
- *What* the actor is to do
- *Time bounds*: how fast, etc
- What value and bounds
- *Rationale*, the description of the requirements role, purpose, motivation, and/or intent must be clear and readable

5.1. PROPERTIES OF A GOOD REQUIREMENT

A well-written requirement exhibits the following characteristics:

- Complete – contains sufficient detail to guide the work of the developer & tester
- Correct – error free, as defined by source material, stakeholders & subject matter experts
- Concise – contains just the needed information, succinctly and easy to understand
- Consistent – does not conflict with any other requirement
- Unambiguous – must have sufficient detail to distinguish from undesired behaviour. includes diagrams, tables, and other elements to enhance understanding
- Verifiable (or testable) – when it can be proved that the requirement was correctly implemented
- Feasible – there is at least one design and implementation for it.
- Necessary – it is traced to a need expressed by customer, user, stakeholder;

- Traceable – can be traced to and from other designs, tests, usage models, etc. These improves impact assessment, schedule/effort estimation, coverage analysis scope management/prioritization

5.2. THE SYNTAX OF REQUIREMENTS

The grammar used to refer to items, properties, actions, conditions, etc. should be consistent among the requirements. It should be rare that there is more than one-way use to express a clause. Below is a specific, stylized syntax for the requirements.[2] The generic syntax for requirements is:

> *The [actor] shall [action] <optional object> <optional trigger> <optional preconditions>*

The table below synopsizes the main elements of the requirement:

actor	the system
action	create
object	an invoice
trigger	When an order is shipped
precondition	and the order is not prepaid

Table 3: *Decomposition of example requirement*

Requirements of the same "type" should employ the same format. Tip: maintain a handbook of templates or examples to represent key (or common) types of requirements. The types of requirements and their typical phrase patterns:

Type	Pattern	Notes
Ubiquitous	The *[actor]* shall *[action]* <*optional object*>	Requirement is always active
Event-driven	The *[actor]* shall *[action]* <*optional object*> when *[trigger]* <*optional precondition*>	Required response to a triggering event
State-driven	The *[actor]* shall *[action]* <*optional object*> while *[state]*	Required response in a specified state
optional	The *[actor]* shall *[action]* <*optional object*> where *[feature included].*	Applicable only if feature is included
unwanted	The *[actor]* shall *[action]* <*optional object*> if *[unwanted condition /event].*	

Table 4: *Summation of the typical requirements patterns.*

5.3. REQUIREMENT FOR UBIQUITOUS BEHAVIOURS

Ubiquitous requirement states a fundamental system property or behaviour. It defines system behaviour or features that must be active or present at all times. This form of requirement has the template:

> *The [actor] shall [action] <optional object>*

For example:

> FOOBAR123: The product shall provide a connector to the mains power.

[2] Based on EARS. EARS has a slightly different normal order. I have kept the subject at the head of the sentence in each case.

Note: This form of requirement is rare. When reviewing, look for missing triggers and other conditions on the requirement.

5.4. REQUIREMENT FOR EVENT-DRIVEN BEHAVIOURS

Event driven requirements are initiated only when a triggering event is detected. The trigger must be something that the system itself can detect. This form of requirement has the template:

> *The [actor] shall [action] <optional object> when [trigger] <optional preconditions>*

For example:

> FOOBAR124: The light driver shall turn the light off if light switch is set to the off position.

See section 4.7 for descriptions of an action,

Notes on the trigger can be:

- A defined (named) event (see section 4.8)
- When a property changes to a value
- When a property changes from a value

Notes on the optional pre-conditions – may need to differentiate state and pre-condition…

5.5. REQUIREMENT FOR STATE-DRIVEN BEHAVIOURS

Requirement is active while the actor is in a defined state

Requirement is "continuous", but only while the system is in the specified state. This form of requirement has the template:

> The *[actor] shall [action] <optional object> while [state]*

Example

See section 4.7 for descriptions of an action,

See section TBD for descriptions of state. Usually an internal, "hidden" state, rather than conditions of properties

5.6. REQUIREMENT FOR OPTIONAL FEATURES

This form of requirement has the template:

> The *[actor] shall [action] <optional object> where [feature included]*

Example

See section 4.7 for descriptions of an action,

5.7. REQUIREMENT FOR UNWANTED BEHAVIOURS

This is a variation of event-driven requirement. The trigger or condition is one that is unwanted, error condition, failure, fault, disturbance, etc. This form of requirement has the template:

> *The [actor] shall [action] <optional object> if [unwanted condition/event]*

Example

See section 4.7 for descriptions of an action,

5.8. COMPLEX REQUIREMENTS

These employ decision tables, and other forms.

6. OTHER WRITING TIPS

For each element in the document: is it the right kind of document? Does it specifies what the product does, how it behaves and performs. Does it specify implementation or design? If so, remove that. {Don't tell me how to do it when you don't know how to do it.}

- Use only a single "shall" and only a single action per requirement
- Do not use adverb disjunctions, or adverb adjuncts.
- Do not use uncomparable adjectives.
- The requirement should clearly specify the actor and what it should do – how it should behave.
- Do not couch a specific implementation as a requirement.

CHAPTER 3

Specification & Requirements Analysis

This chapter describes how to analyze a specification and its requirements:

- Overview
- Definition of terms and phrases
- What to look for
- Signal processing
- Requirements
- How to decompose and organize the information

7. OVERVIEW

The analysis is intended to identify and organize the definitions of terms and key phrases:

- The nouns and actors – that is, the kinds and instances
- The properties of the nouns, and their allowed values and constraints
- The states and enumerations
- The key, well-known events
- The actions – the definitions of actions in specific cases
- The conditions

7.1. STEPS IN THE ANALYSIS PROCESS

The steps in the process of analyzing requirements is:

1. Identify the *kinds* of things being referred to (or discussed)
2. Identify the sets of *properties* that an entity may have
3. Identify the set of *acceptable property values* for each property
4. Identify the *states* that are composed of or identified by property values
5. Identify the *constraints* on the property values
6. Identify the *events* that may occur

7. Identify and define the *actions* that may occur

The process of breaking the process out into small steps is to:

1. Manage complexity
2. Support test /debug that conditions are met
3. Systematic mechanical rules to go from requirements to implicit.

8. DEFINITIONS OF TERMS AND PHRASES

This will define the nouns and their modifiers:

- Things – e.g. the actors
- The kinds of things
- The properties that things (and kinds of things) may have
- The values that properties may have
- How to compare properties and values
- The derived properties

Scan the source documents and build up a lexicon. The development in this section parallels section X in chapter 2. The source documents includes

- Requirements specifications for this subsystem,
- Requirements documents for each of the larger systems that it is part of
- Top-level design documents for the electrical function of this subsystem (if one exists)
- Top-level design documents for the larger system that this is part of
- Documents refer to by each of the above.

8.1. DEFINITIONS REVIEW

Reviewing should look to identify:

- Is the definition clearly demarcated?
- Is the definition duplicated? Do the definitions conflict?

8.2. KINDS AND INSTANCES

Go thru and build table of what kind of things each kind is. Note: see the appendix for a common stock of kinds of things.

***Table 5:** Kind inheritance hierarchy*

Kind	Kind of	Description
yellow LED	LED	A kind of LED used for status

Instance	Kind of	Description
Status LED	yellow LED	A kind of LED used for status
CAN interrupt handler	interrupt handler	
DMA		

Table 6: Instance kind-of and description

Common nouns are kinds;
Proper nouns are often instances.

common nouns
proper nouns

An instance can be a "kind" of several things; a kind can be a kind of several other kinds of things too.

The *ancestors* of a kind or instance is the set of all kinds of things it may be… this includes all of the ancestors of those kinds.

ancestor

Check:

- Is each instance or kind referred to in a consistent, uniform manner?
- Are each of the ancestor kinds known – are there some that are undefined?

8.3. PROPERTY DEFINITIONS AND POSSIBLE VALUES

Go thru the specification and build a table of the properties. Note: see the appendix for a common stock of dimensions and their units.

Property	Kind of	Dimension	Possible values
brightness	number	lumens	
gender			{male, female, neuter}

Table 7: Property type definitions

The set of values that a property can take on are usually one of:

- Categorical (aka nominal) properties include a set of values it can take on,
- A scalar property, these often have a dimension (which may it's own a preferred units). These may also have a set of values that it may take on; usually they are range of allowed values.

Check:

- Is the property is referred to in a consistent, uniform manner?
- Is the value kind and possible values suitable for the given dimension? For instance, a dimension of "length" cannot meaningfully take on possible values of "red, male, squirrel"

8.4. WHICH PROPERTIES A KIND OR INSTANCE MAY HAVE

Create a table of the properties kinds and instances have.

Kind / Instance	Properties
yellow LED	brightness

Table 8: *The properties for kinds and instances*

A property should be defined only once a given 'kind' hierarchy:

- If the property is defined for both an instance, and its kind (or one of its ancestor kinds)… it should be defined only for the kind.
- If the property is defined for a kind and for one of the kind's ancestors, it should be defined only in the ancestor.

An instance or kind may have anonymous, unnamed properties. Fill in the table:

Kind/Instance	Kind of	Dimension	Possible values
LED			{on, off}

Table 9: *Anonymous property type definitions*

If a set of possible values is identical to that of any of the property the instance, kind or ancestor has, investigate if that property should be used.

8.5. PROPERTY VALUES

An instance or kind may further specify the range of values for a property, or even a specific value for the property:

Kind/Instance	Property	Possible values
image	width	240 pixels
	height	216 pixels
UART	data rate	9600bps .. 115000bps
	stop bits	1
main board	operating current	≤130mA
	operating voltage	3.3v .. 6v

Table 10: *Anonymous property type definitions*

If a set of possible values is identical to that of any of the property the instance, kind or ancestor has, investigate if that property should be used.

8.6. COMPARISON: COMPARATIVES, COMPARABLES, AND SUPERLATIVES

Look for the definition (often implicit) of comparable or comparative adjectives, and build a table of these relations. For properties, this can be a regular change in affix from the base property name. (See the appendix X for a table of common comparable and comparatives). If

the text is defining a comparable or comparative, look for the comparison relation. The comparison function should match the underlying domain.

Adjective	Property	Comparison
brighter	brightness	>

Table 11: *Comparative definitions*

A comparable adjective should define a value set to compare against. Note: a comparable may define itself based on the comparison relation, or in terms of a comparative. In the case of the later, use the property and comparison relation from the comparative.

Adjective	Property	Comparison	Auxiliary
brighter	brightness	>	
bright	brightness	>	80

Table 12: *Comparable definitions*

In some cases it is useful to define comparable adjectives on a kind or instance.

Kind/instance	Adjective	Property	Comparison	Auxiliary
	brighter	brightness	>	
	bright	brightness	>	80

Table 13: *Comparable definitions for an instance*

Check:

- Is the comparison relation supported by the underlying domain (kind) of the property?
- Do comparables have a value set defined to compare against?

8.7. SUPERLATIVES

Superlatives select a single item (or set) from a set of candidates. Look for the definition (often implicit) of superlative relations, and build a table of these. For properties, this can be a regular change in affix from the base property name, or comparative adjectives. (See the appendix TBD for a table of common superlatives).

Adjective	Property	Comparison
brightest	brightness	>

Table 14: *Superlative definitions*

Note: a superlative may define itself based on the comparison relation, or in terms of a comparative. In the case of the later, use the property and comparison relation from the comparative.

8.8. DERIVED PROPERTY: MAPPING TO STATES OF PROPERTIES TO AN ENUMERATED VALUE

Look for definitions mapping of property values or subsystem states to so some state of the system definition and build tables of these. (This can be thought of as defining a property in terms of more fundamental properties.)

Table 15: *How classify state*

Inputs				**Outcome**	**Description**
v1	v2	v4	v4		
state	**etc**	**off**	**1..3**	state	

Define the inputs to the classification; these can be the name of any earlier defined property, or subsystem state. Fill in the input cells with sets of acceptable values for that rule. And the outcome that combination maps to.

Check:

- Check that the input values are in the domain (value set) for their respective properties.
- Check that the outcome values in the value set for the property.
- Variables are not listed twice
- The outcome variable is not also an input at the same table; The *previous* value of the outcome variable may be used as an input.
- Ambiguity: Can the inputs be interpreted different ways? Is there sufficient detail to distinguish one outcome state from another? Are multiple outcomes selected by the same input?
- Vagueness: Are there combinations of inputs that do not have a defined outcome?
- Acyclic: the input variables do not refer to current-time-step value of a state/property that directly or indirectly derives from the current-time step value of this state classification.

8.9. CONSTRAINTS

Table of what property values rules. These constitute acceptable states of the system.

Table 16: *Description of constraints*

Expression	**Relation comparison expression**	**Description**

Check:

- Is the constraint tautologically true or false?

- Is there a conflict between the constraints?

8.10. EVENTS

Look for definitions of key, named events. (See the appendix TBD for a table of common well-known events). These have a name, rather than "property X taking on value Y." For instance: start up, power on reset, out of seat, engine start, engine kill.

Instance/Kind	Event	Description
yellow LED	blink	

Table 17: *Description of events*

Check:

- Are the events something that the system can detect?

8.11. ACTIONS

Look for definitions of key actions. (See the appendix X for a table of common well-known actions).

Instance/Kind	action	Description
yellow LED	blink	

Table 18: *Description of actions*

Identify

- The actors. These should be instances (not all instances will be actors)
- The item acted upon (optional). This should be an instance of the hardware
- The actions

Actors and the set of actions that it can take. The set of states that a thing can be in.

Actions:

- The outcome, such as the state of a property or multiple properties
- Value setting of a noun
- Specific steps. Each step is also an action. Check for circular definition

Identify the outcomes of the action

Identify the performance requirements, the time points of the action: how fast?

Check:

- Are the actions something that can be accomplished?
- Are the actions testable?
- What is the outcome of the action? Is the outcome testable?

8.12. CROSS REFERENCE TABLES

It is helpful to the analysis to construct cross reference tables:

- Map kind and instances to the set of all ancestor kinds
- Map property name to instances/kinds holding it;
- Map possible value to the property name (table X), or state classification table
- Map name to event definition / description.
- Map possible value to instance anonymous property
- Map action names to action definitions

9. SIGNAL PROCESSING

9.1. FILTER SPECIFICATION

Depending on the type of filter specified which parameters are needed. Identify the type of filter being specified and that it has defined its key parameters. The types of filters supported include:

***Table 19:** filter parameters*

Filter	Parameters to specify
Band-pass filter	low cut-off frequency and high cut-off frequency
Band-stop filter	see *notch filter*
DC removal	Aggressiveness coefficient
Equalizer	
Exponential smoothing	Smoothing coefficient
High-pass filter	cut-off frequency
High-shelf filter	shelf frequency
Low-pass filter	cut-off frequency
Low-shelf filter	shelf frequency
Notch filter	low cut-off frequency, high cut-off frequency
Peak filters	peak frequency
PID control loops	Max overshoot
Spike removal filter	

9.2. SIGNAL PROCESSING SPECIFICATION REVIEW

Check that the filters have specifications for all of the relevant parameters.

9.3. CONTROL LOOPS

Some quality characteristics to define for a control loop:

- Max ripple
- Max overshoot

- Response time
- Max error

10. ANALYZING REQUIREMENTS

There are three top level parts

- The trigger
- An optional time between the trigger and initiating the action. Such a timer is usually cancellable.
- The action to carry out

The trigger can be

- A Boolean expression on a property, and/or
- The start of another action, and/or
- The end of another action

Note: some implicit events:

- Named events (see section XLINK)
- When a property or state changes to a value
- When a property or state changes from a value
- When an action starts
- When an action ends
- What value and bounds
- Rationale, the description must be clear and readable

Trigger	Delay	Action	Description

Table 20: Requirement link

CHAPTER 4

Requirements Checklists

This chapter summarizes the requirements review checklists

- Requirements review checklist

11. REQUIREMENTS REVIEW CHECKLIST

See also

- Appendix G for the *Code Complete* Requirements Review check lists

Names:

- ☐ Are the names clear and well chosen? Do the names convey their intent? Are they relevant to their functionality?
- ☐ Do they use a good group / naming convention (e.g. related items should be grouped by name)
- ☐ Is the name format consistent?
- ☐ Names only employ alphanumeric characteristics?
- ☐ Are there typos in the names?

11.1. ARE THE PROPERTIES, STATES AND ACTIONS WELL DEFINED?

- ☐ Is a definition duplicated?
- ☐ Is a property defined multiple different times.. but defined differently?
- ☐ Are the definitions complete?
  - Are all instances and kinds defined – or some missing?
  - Are there undefined (i.e., referred to, but not defined) nouns, properties, verbs?
  - Are events referred to but not defined?
- ☐ Are they consistent?
- ☐ Are the properties something that the system can measure or otherwise detect?
- ☐ Are the instances something that the system can identify or otherwise distinguish?
- ☐ Is a state not needed? Is it unused by any state classification, action, event, or requirement?
- ☐ Is a property not needed? Is it unused by any state classification, action, event, or requirement?
- ☐ Are the properties something that the system can detect?

- ☐ Are the events something that the system can detect?

11.2. REQUIREMENTS REVIEW

Reviewing requirements should look to identify:

- ☐ Are the requirements organized in a logical and accessible way?
- ☐ Is the requirement clearly demarcated?
- ☐ Does the requirement have a clear and fixed identifier? Is the identifier unique?
- ☐ Is the description supporting the requirement clear? Is it sufficient to support the requirement?
- ☐ Is the requirement too wordy? A requirement should be concise, containing just the needed information.
- ☐ Does the requirement use the proper modal auxiliaries?
- ☐ Does the requirement have the right conditions? The ubiquitous form of requirement is rare. Look for missing triggers and other conditions on the requirement.
- ☐ Are the time-critical features, functions and behaviours identified? Is the timing criteria specified?
- ☐ Is there requirement declarative? Or is the requirement an attempt to repackage an existing implementation with imperative statements? These are bad.
- ☐ Does the requirement conflict with any other requirement? Is its use of conditions (e.g. thresholds) consistent with the other requirements?
- ☐ Is the action to carry out clear? Is the action well defined within the rest of the specification?
- ☐ Are the actions something that can be accomplished?
- ☐ Duplicated requirements?
- ☐ Ambiguity. Can the requirement be interpreted different ways? Is there sufficient detail to distinguish from undesired behaviour?
- ☐ Is the requirement vague or ambiguous in any way? Pronouns, demonstratives, and indexicals often introduce ambiguity.
- ☐ Is the requirement specifying a single action.. or many? A requirement should specify only a single action.
- ☐ Complexity. Is the requirement over specified, too complex?
- ☐ Requirements that are too expensive, burdensome, impractical or impossible
- ☐ Are the requirements ones that fit the practical use with customer wants/needs/etc?
- ☐ Is the requirement unnecessary? Does it lack a trace to a need expressed by customer, user, or stakeholder? Is each requirement traceable to a customer that requires it?
- ☐ Check for consistency and sufficient definition
- ☐ Does the requirement have errors, such as misstating bounds, or conditions in the source material, or from other stakeholders or subject matter experts?
- ☐ Are there missing requirements? Is there a lack of sufficient detail to guide the work?

11.3. ARE THE REQUIREMENTS TESTABLE

- ☐ Are the triggers something that the system can detect?
- ☐ Is the action or result of the requirement observable? Can it be measured?
- ☐ Are the quality requirements measurable?

- ☐ Is the requirement time bound? Is there a clear time bounds between the condition or trigger, and the action?
- ☐ Is the requirement untestable? Is there a direct means of stating how to test that the requirement was correctly implemented?
- ☐ Is the actor to carry out or meet the requirement clear? Is the actor well-defined within the rest of the specification?
- ☐ Are the actions testable? Is their outcome testable?
- ☐ Is the requirement bounded? Or is the actor allowed to do the requirement at the end of the universe?

11.4. THE LEADS REVIEW REQUIREMENTS:

- ☐ Are they complete? Are requirements or definitions missing? Are there undefined nouns, properties, verbs?
- ☐ Are they consistent?
- ☐ Are they doable?

CHAPTER 5

Electronics design description outline

This chapter describes the preferred outline for an electronics design description:

- Main outline
- Synopsis & Front matter
- Overview
- Detailed design
- Power characteristics
- Trim and calibration
- Connectors & Pin Maps

12. MAIN OUTLINE

This chapter describes my preferred approach to writing a design description. The role of a design description is to communicate with future electronics maintenance, software and test teammates; and to reduce the puzzles and mysteries in being handed a completed electronic design and being expected to make it work/modify it/test it. The following is the outline for a design description:

8. Synopsis
9. Other front matter. These, if part of a larger document, should be placed in the preface or appendices of the larger work:
    a. Glossary, acronyms
    b. Related documents (documents that are part of the product)
    c. References, resources, suggested reading
10. Overview
    a. The list of features that the electronics is responsible for
    b. A basic block diagram of the electronics designs organization, calling out key items referred to in the document
11. Detailed design
    a. Detailed block diagram of the electronics designs organization. This should include the connectors, power management, sensors, drivers, microcontrollers and other subsections that will be described in detail in the rest of the document.

b. Module (physical) connectors
c. Clocks
d. Sensors, the signal chain and other inputs to the microcontrollers
e. Safety / self-protect circuitry. This includes reset supervisors
f. Driver circuit(s)
g. Microcontroller

12. The power systems, distribution, characteristics.
13. System model / equations for key modes. How to perform specific operations & tasks. Control equations. Analysis to support the design or requirements.
14. Pin-maps

13. SYNOPSIS AND FRONT MATTER

THE SYNOPSIS. A one or two paragraph synopsis of what the modules is and its role in the product is.

THE RELATED DOCUMENTS, SPECIFICATIONS. The documents to list internal organization & project standards, and design specifications. Include a designator for each document. Use this through the remainder of this specification to refer to the document.

THE REFERENCES, RESOURCES, SUGGESTED READING. The documents to list include data sheets, industry and legal standards, communication protocols, etc. Include a designator for each document. Use this through the remainder of this specification to refer to the document.

THE ACRONYM AND GLOSSARY TABLES. Define all acronyms, terms and phrases.

14. DESIGN OVERVIEW

Describe the role and responsibility of the electronics. Include the features that it is responsible for.

- Overview
- Power sources
- Physical electrical connections
- Test pads
- Calibration

Include a diagram summarizing the electrical design, with the major sections and their interconnections. This may include a reference to external elements that it controls or depends on.

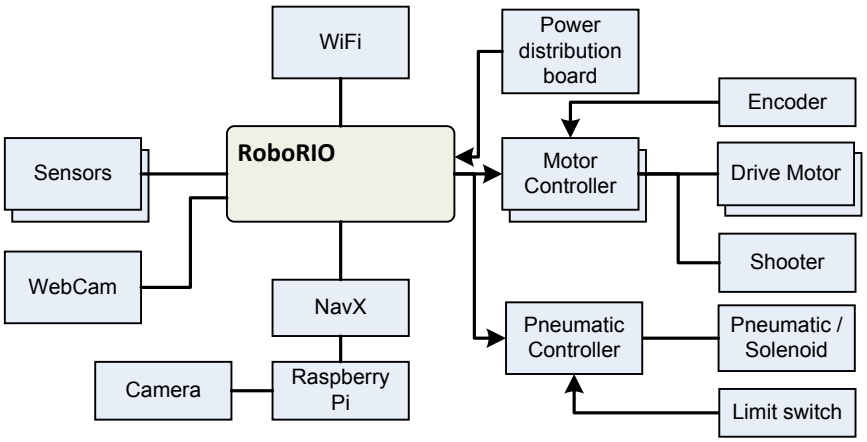

Figure 12: *Elements of the design as it fits in the overall system*

Note: this is an illustrative diagram, but won't be thematically

Provide a description of the system elements external to the electronics design:

External element	Description
element 1	Description of the element
…	
element n	Description of the element

Table 21: *The elements external to the electronics design*

Provide a description of the main elements of the electronic design:

Element	Description
element 1	Description of the element
…	
element n	Description of the element

Table 22: *The electronic design elements*

14.1. TIP: A CATALOG OF COMMON ELEMENTS/SUB-MODULES

I recommend employing a "recipe book" or catalog of common design elements. The catalog should be organized by function – that is, grouping related modules together. The designs (recipe) should have:

- A unique identifier for the design (to distinguish between them). The identifier should indicate the function group of the design
- A reference schematic
- Potted, reusable text describing the role, function, behaviour, etc of the module

14.2. NOTATION & NAMES

A good notation for naming eases understanding of how the sub-products work together. The following areas should use a related naming notation:

- Electronic modules / subsystems
- EE schematic wires/pins
- Software modules
- File names
- Procedure/function names
- Type names
- Variable names

My preferred order is (left to right) *Module kind*, *instance identifier*, *role* & *driver* & *polarity*. These elements should separated by underscores. The notation elements to distinguish a particular module would look like:

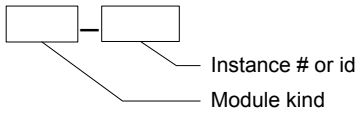

Figure 13: *Module identifier notation*

- The module kind; the table(s) of abbreviations and acronyms is a good place start for identifiers for kinds of modules.
- A means to distinguish between different instances of the module, if there is (or may be) more than one instance. This may be a number or alphanumeric designator.

Tip: do not depend on capitalization. A name should have a suggested or preferred capitalization. However, two different items must have names that differ in more than just the capitalization. For instance, "RESET" and "Reset" must refer to the same item (e.g. signal).

The signal lines should be identified with a notation such as:

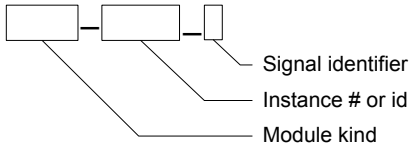

Figure 14: *Signal identifier notation*

Other elements of the identifier can optionally indicate:

- The role e.g. data, chip select, clock
- The driver of the pin, usually relative to the microcontroller:
- The polarity: active high, active low, open drain, open collector

14.3. POWER SOURCE

Describe the intended energy source of the module. Is energy applied all of the time? What is the envelope of power that it can support? Does it have user serviceable batteries?

Are there expectations on the user?

- Does it need to be recharged?
- Is the operator expected to regularly change the batteries?

14.4. PHYSICAL ELECTRICAL CONNECTIONS

This section provides summarizes each the connections to motor driver. There connectors providing power, communication, sensors and tuning are:

- Battery power connector
- Battery communication connector
- Diagnostic port(s)
- Manufacturing test connector
- Programming/Debugging port
- UART connector(s)

Describe the connectors providing power, communication, etc

Table of the connectors

Describe each connector (an example section is below)

14.4.1 Connector description

Type of connector

[diagram Schematic of the interface]

Figure 15: *Connector ABC and it's interface*

Connector identifier (see notation above)

Connector pin identifiers (see notation above)

Signal to reset of schematic (see notation above)

[table]

14.4.2 Manufacturing test connector

The electrical design will provide test points and/or test access sufficient to support design verification, manufacturing operations and manufacturing test. This includes a connector for manufacturing test & debugging.

The connector includes an [TBD, uart?] interface. These are used for testing the module, and to configure it. See [tbd xlink] for a discussion of manufacture time testing. For manufacturing test, the connector footprint will engaged with a test fixture. In a development environment, an adapter cable (available from [TBD]) will be used to connect to debugging tools. The footprint is shown below.

[diagram connector footprint]

Figure 16: *Foot print for the manufacturing connector*

Below shows a shortened, representative, image of the cable. A second adapter will be used to mate it with any debugging or desk programming tools:

[diagram cable]

Figure 17: *Debug cable*

The connector that mates with the circuit board looks like:

[diagram mating connector]

Figure 18: *Detailed view of the connector pins*

14.5. TEST PADS

The electrical design should provide test points and/or test access sufficient to support development, design verification, and manufacturing test. Describe the pads, their intended role.

See section 14.2 for suggestions in labeling the test pads.

14.5.1 Dog-bones / severable test points

Dog-bones may be employed to remove optional circuits, allow measuring current flows, injecting signals or allow testing signal failures.

14.6. CALIBRATION

Will there calibration be needed in the product? Is per design, or per unit? What aspects will need to be calibrate? For instance:

- Crystal trim
- Temperature coefficients

Who will trim or calibrate these? Will they be stored? Where will they be stored?

15. DETAILED DESIGN

This section delves into more detail regarding the electronic design. Include a diagram summarizing the design, with the major sections and their interconnections. This may include a reference to external elements that it controls or depends on. A typical block diagram looks like:

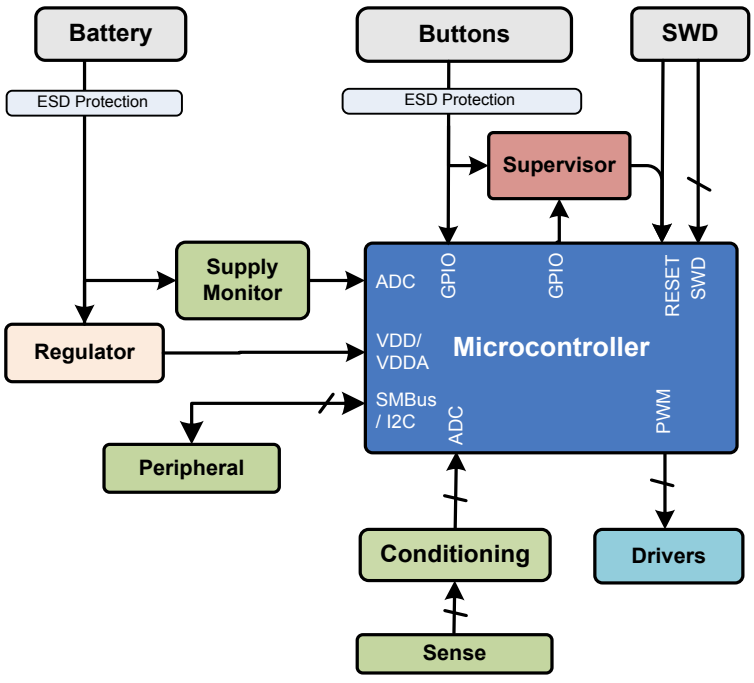

Figure 19: *Detailed block diagram of the product*

The major sections of the design include:

- The electrical connectors are in the top row (or left column or …).

- The power management is in the left column (or bottom row or …).
- The microcontroller is in {color}.
- The sensors are in {color}.
- The motor driver is in {color}.
- The supervisors – e.g. short/stall/over-current detection – are in {color}.

Plan to discuss:

- Power source, regulation, protection
- Sensors, signal conditioning
- Outputs, drivers e.g. relays, smart FETs, etc.
- Microcontroller
- Communication
- Data Storage
- Trim and calibration
- Temperature operating range

15.1. POWER SOURCE & REGULATION

This section is where the designer describes how energy is:

- Stored,
- Replenished,
- Distributed to the electronic elements, and the external elements,
- How the energy is converted to a useful form. How it is converted to internal forms of power; how it is applied to work. Note: often the work is done by elements that the electronics support, but a discussion of this is still mandated.
- How the energy is managed:
  - How the capacity is measured
  - How the flow and/or utilization is measured
  - How it is enabled and cut-off
- The limits of the system:
  - The maximum energy that can be stored,
  - The maximum flow,
  - Limits to power distribution

15.1.1 Power distribution tree

Designs will include some method to distribute electrical power to the components. Describe the power distribution:

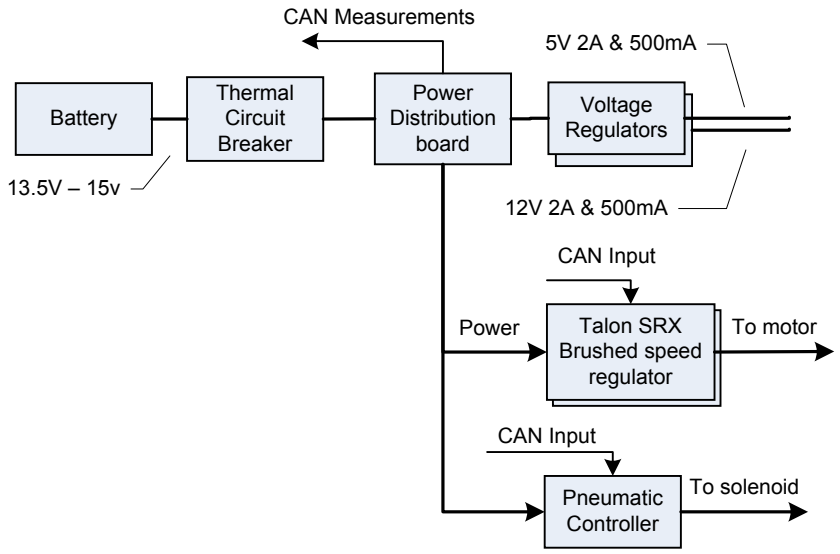

Figure 20: *Power distribution diagram*

Provide a description of the main elements of the electronic design:

Element	Description
Battery	The energy is supplied by a rechargeable Lithium Ion battery.
Pneumatic controller	Used to drive the solenoid to release air from the pneumatic tanks. The release is enabled by a CAN command.
Power distribution board	Distributes power from the battery. It has individual fuses, and reports – via CAN – the current flowing thru.
Talon SRX brushed speed regulator	Used to run the brushed motors. It may take a feedback from an optional encoder. The speed is set by a CAN command.
Thermal circuit breaker	This is a circuit breaker used to protect the battery, wiring and electronics in the event of a short circuit or overload condition.
Voltage regulators	The regulators ensure that a specific, fixed voltage is applied despite the changing battery voltage.

Table 23: *The electronic design elements*

15.1.2 Power source

How is the energy – or power – to be provided to electronics? Does it have a part number, and/or specification? What are its characteristics, such as its voltage range, and capacity? Is there an impedance to the power source?

Does the battery have an impedance or other notable characteristics? Include an equivalent circuit model where possible.

Is the electronics (and/or software) to monitor or condition the power source in any way?

Are there start-up or operating requirements? Such as not exceeding transition power while starting up (to prevent brown-outs)?

Are there safety or other quality requirements in using the power source?

15.1.3 Power decoupling capacitors

To accommodate this high internal resistance, a Vcc bypass capacitor is used. This capacitor will act as the primary source of power during these periods. A 10pF bypass capacitor is used as well, to shunt the RF & other energy picked up by the battery and traces.

[etc]

15.1.4 Reverse battery and other protections

Is there a fuse?

Are there protections against reverse battery connection? voltage applied to connector signals? “load dump” protection?

15.1.5 Power regulators

Boost, buck, buck-boost, LDOs.

Are there start-up or operating requirements? Such as not exceeding transition power while starting up (to prevent brown-outs)?

Is there is a low power setting? Is there safety requirements around the power regulators? Is there a power tree? What is the topology? Are there rules to enabling and cutting off (or disabling) power?

Are there special limits / configurations for the regulators? For instance, buck and boost regulators often emit unacceptably unless configured to constraint their frequency emissions.

With variable or controlled power supplies, describe how they will be set and/or controlled. Provide closed form equations to map the target voltage (or power) to the settings.

15.1.6 Measurement

Measures

Energy store, supply

- state of charge

- voltage

- impedance

Power regulator

- voltage measurement

  * brown out detect

- current measurement

Loads

- current esp at key points

- voltage measure, e.g. to measure impedance

15.2. SENSORS AND OTHER INPUTS

Summarize the signal lines and sensors into the system:

- Signal one
- Six signals of another
- Power source measurement
- Power regulator measurement
- Temperature measurement
- etc

15.2.1 Layout

Layout can impact the quality of ADC measurements. Make note of the trace placement requirements, ground plane requirements, placement of filter capacitors, etc.

15.2.2 Signal conditioning for ADC inputs

This section describes the signaling conditioning used to prepare signals for microcontroller ADC inputs. This section should identify ratios, pre-conditions, and other information to support section 15.4.7. The following signal conditioning circuit to prepare the signal for the microcontroller ADC inputs:

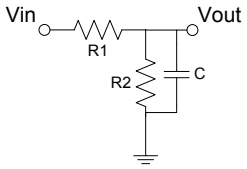

Figure 21: *Resistive-divider*

The divider reduces the voltage input range to one that can be measured by the microcontroller's ADC inputs. The equation for the V_{in} to V_{out} is

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2}$$

Equation 1*: The voltage ratio for a resistive-divider*

The equation for the time constant, in the resistive-divider filter topology, is:

$$\tau_{divider} = C \frac{R_1 R_2}{R_1 + R_2}$$

Equation 2*: The time constant for the resistive-divider filter*

Note: if using an impedance divider – such as a capacitive divider – describe very carefully how to use, time the measurements, and calibrate it.

Summarize the characteristic parameters of the conditioning and filter. Resistive-divider filters have the following characteristic parameters:

Parameter	Value
Divider ratio (nominal)	6.63 : 1
Output impedance	*low (<500Ω)*
Time constant ($\tau_{divider}$)	$1.35 \cdot 10^{-6}$

Table 24: *Resistive-divider characteristic parameters*

Does the setting settling time require software consideration? Is the settling time compatible or in conflict with ADC signal oversampling techniques? The ADC inputs are discussed in a later section.

15.2.3 Power source measurement

Is the power source measured? Cross-checked?

Describe what supply voltages are measured, the expected & acceptable ranges, tolerances and the voltage cross checks.

Describe the signal change from the power source to ADC (or other) input.

15.2.4 Power regulator measurement

Are the power regulator voltages measured? Many microcontrollers include a feature to measure its supply, using internal precision voltage source. By measuring this and the regulator supply voltage, [TBD] a crosschecks the ADC reference voltage.

Describe what supply voltages are measured, the expected & acceptable ranges, tolerances and the voltage cross checks.

Describe the signal change from the power source to ADC (or other) input.

15.2.5 Temperature measure

Many microcontrollers and sensors include a feature to measure its internal temperature.

This can be tested by having the device firmware report the temperature and comparing against a temperature probe on the manufacturing line. It has a per unit calibration value.

15.3. OUTPUTS AND CONTROLS

TODO describe the outputs,

TODO describe what it is controlling. Describe how they will be set and/or controlled. Provide closed form equations to map the target (or power) to the settings to apply. If a dynamical system of equations is needed, provide that.

For example:

- Is it driving a motor?
- Output relays?
- "Smart" FETs?

15.3.1 Relay

[describe]

15.3.2 H-bridge driven outputs

[describe]

15.3.3 Smart FET driven outputs

[describe]

15.4. MICROCONTROLLER & SOC/SIP

Introduce the microcontroller or SOC/SIP module here. Who is the manufacturer, and what is model identifier?

[Provide a diagram of the microcontroller boiler plate]

Figure 22: *The microcontroller power, and basic I/O*

The microcontroller is responsible for [describe]. The section describes the microcontroller support, and its several special features:

- The digital power supplies
- The analog power supplies
- The microcontroller's clock(s) and clock failure detect
- Analog to Digital Converters
- Digital Inputs & Outputs
- Digital to Analog Converters
- SPI interfaces
- I2C interfaces
- UART
- Debugging interfaces
- The microcontroller's pin map, organized by function grouping

15.4.1 The Digital Power supplies

The power supplies for the microcontroller core, the GPIO in/output pins, and battery backed up region are supplied by the [TBD] regulator (see section {xlink}). Several microcontrollers allow a separate power supply to the core (which often may run at a lower voltage and power consumption) from the GPIO outputs (which may supply a higher voltage, and often has higher current demands).

[diagram]

Figure 23: *The power supply for the microcontroller core and battery backed up region*

[diagram]

Figure 24: *The power supply for the microcontroller digital IO*

The smaller (bypass) capacitors should be located as close to the microcontroller pins as possible; the supply capacitors should be located close to those capacitors. These serve as noise suppression capacitors, taking off any noise picked up by the trace to the regulator. Effectively the trace is an inductor and, with this capacitor, it forms a low pass filter.

15.4.2 The Analog Power supplies

The power supplies for the microcontrollers analog to digital converter, comparator, and other analog domains are supplied by the TBD regulator (see section {xlink}).

[diagram]

Figure 25: *The power supply for the analog to digital converter*

Analog power supply can impact the quality of ADC measurements. Make note of the power regulator requirements, placement of filter capacitors, trace placement, separation of grounds, etc.

Data sheets have recommended noise bypass, power filtering, etc.

15.4.3 Operating modes

Are there notable operating modes of the microcontroller / module? Describe them

15.4.4 Microcontroller startup

Describe anything special: Boot pins, external program load (it isn't intended to describe our design of software.. only how the received microcontroller or module starts up)

15.4.5 Clocks

How many clocks/oscillators support the microcontroller? What is the accuracy of the oscillator (± TBD ppm)?

Are there mechanism that detects failures with the high-speed clock? Describe them, and where to find more information. What happens if a failure is detected?

15.4.6 EEPROM Interface

Are there connections to EEPROM or other external non-volatile storage?

15.4.7 ADC: Analog (Linear) to Digital Converters

This section provides key information to configure the ADC (and software that uses it) based on the electronic design. Find the impedance of the input (for the signal chain), the number of converter bits, the ADC impedance, and compute:

- The minimum sample time
- The max sample rate
- The tau roll off of a signal – how fast does the signal to the ADC respond to the source input signal?

15.4.7.1 Minimum sample time

The settling time get enough of the signal from the source into the last bit of the ADC.

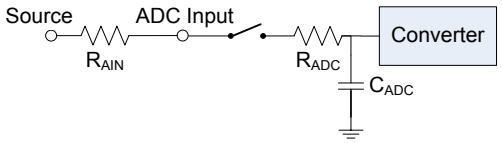

Figure 26: *Equivalent RC circuit for ADC input*

The ADC actions that [intro]

1. An ADC typically closes the switch in the above diagram, allows the sample capacitor discharge/charge so that its voltage matches the ADC input voltage. This duration is the ADC *sampling time* (also called *acquisition time*).
2. The switch is opened, and the converter maps the voltage to a digital value. This duration is the *hold time* (also called *settling time)*.
3. The sum of these two durations is the *conversion time* (or *latency)*.

The ADC sampling time should be set to a time greater than the RC time constant of the above circuit scaled by the number bits to resolve. A typical ADC can be configured for 6 to

12 bits of resolution per sample. With a 12 bit ADC, a sampling time of 8.32 tau *or greater*[3] should be used. The minimum sampling time as a function of the time constant and ADC resolution is:

$$sampling\ time \geq \tau_{ADC} \cdot nbits \ln 2$$

***Equation 3**: The ADC sampling time*

The time constant, once the switch in the above diagram is close, is:

$$\tau_{ADC} = (R_{AIN} + R_{ADC})C_{ADC}$$

***Equation 4**: The time constant for the ADC input*

With a successive approximation (SAR) ADC, the ADC acquisition time is an integer power of two number of ADC clock cycles:

$$settling\ time = 1 + nbits\ cycles$$

***Equation 5**: The ADC conversion time*

15.4.7.2 Max Sample Rate and Oversampling

The capacitor and impedance of the divider and filter (if any) may limit how frequently the input may be sampled. The max source frequency[4] for an ADC is:

$$\max(f) < \frac{1}{\tau_{ADC} \cdot nbits \ln 2 + settling\ time}$$

$$\max(f) < \frac{1}{\tau_{divider}}$$

***Equation 6**: The ADC's max input frequency*

The ADC may also be employed in an "over sampling" manner. The time between samples must be much less than the rate at which the input signal changes. Oversampling works by measuring the same true signal, plus a small amount of noise. The noise should evenly distribute (on successive samples) the lower ADC bits above and below the true value. By averaging the result, an extra bit of resolution can be obtained.

The output of the sampling and filtering (if any) should be examined on a time scale longer than the settling time from the input to the end of the signal chain. The chain "slows" the signal progression.

15.4.7.3 Calibration and Other issues

Determine if the following are of a concern and/or should be applied:

- ADC offset
- ADC gain
- ADC linearity
- ADC reference calibration

Is ADC characterization/calibration/compensation needed? These are typically not of concern with simple inputs, such as switches. If needed, how will these be accomplished? Where & when the steps be performed?

15.4.7.4 Self-Test and Hardware Test

The external ADC reference can be checked periodically. This can be done by measuring the external reference against the internal reference.

---

[3] For instance, ST micro often recommends 10 tau for time varying signals
[4] Again, ST micro recommends using 1/(10 tau)

The ADC – and signal chain – can be tested by injecting a step signal and checking signal propagation for response:

1. Connect probe to input test points
2. Connect probe to output test points
3. Route to ADC input to DAC in software
4. Inject step pulse into the input
5. Watch the input test points, output test points and the DAC output for the step response.

15.4.8 Digital Inputs and Outputs

Pins that are stuck high or stuck low can be performed by having software individually drive each pin high and low, and checking that the associated test point is driven properly. The microcontroller should not reset during this test.

To check for shorted pins, during the test above have all pins other than the one under test is configured as an input. Then before and after the pin test, check that no other pins are change state, or are in an invalid state as result of the pin-under-test changing state.

15.4.9 SPI

Are the SPI features of the microcontroller employed? What is it used to communicate with? This includes a clock, data to peripheral, data to master, and chip select. The SPI continuity can be tested as follows:

- The SPI pins should be tested as digital input/output pins first. (The microcontroller supports this.).
- The software can directed to perform a long SPI action, with a characteristic MOSI data pattern. The SPI clock and data output line can examined with an external lab tool to check that the edge rise, number of edges, and timing is as expected. It is recommended that the Chip select to the slave peripheral not be employed with the SPI at this time. This checks that the SPI output matches expected patterns and is likely to be interpreted as intended by the slave peripheral.
- The Chip select, is a GPIO line and is tested per the digital in/out tests earlier.

15.4.10 UART Interface

Is there a UART communication interface? Describe it. Does it have handshaking? Is it bi-directional?

***Table 25:** UART pin map*

Port	Pin	Label		Description
		UART_TX	Out	UART from microcontroller
		USART_RX	In	UART to controller

15.4.11 Debugging interface

[describe]

Port	Pin	Label		Description
GPIOA	13	SWDIO	io	Single wire debug IO
GPIOA	14	SWDCLK	in	Single wire debug clock

Table 26: *Single wire debug pin map*

15.5. COMMUNICATION AND RADIATORS

What is the connectivity to the external world? UART, SPI, Bluetooth LE, etc?

Are there transmitters? Are there "wireless" energy transfers? Does the module have other radiations?

15.6. NON-VOLATILE STORAGE

DATA STORAGE. The module/microcontroller/etc includes a non-volatile storage (Flash) to hold data and the program. The microcontroller executes the program directly from the flash. If there are protections available to prevent the areas of flash used for the program from being erased or written, describe them. The program area can also be regularly checked to detect unintended alteration, and loss of integrity from the flash media. Describe these.

The non-volatile storage can tested:

- Check that the storage area is able to hold the range of values. That is, each bit in the storage area can clear and set, that setting or clearing a bit does not clear or set other bits in the storage area. (This is a test that the storage area works as intended, not that the access is done on a bit level.)
- Check that the storage area is non-volatile – that it retains the intended values after power has been removed from the system. This is done by setting the values in non-volatile area to non-default values, removing power (for a time longer than it takes internal power caps to deplete), then applying power, and checking that the storage area holds the expected values.

There is, at present, no means to verify the integrity of RAM against corruption such as from single event upset. Working data held in RAM, including some critical values, is sensitive to such faults. Will the product be expected to operate continuously for years?

The software may employ checks on critical data to mitigate this concern. Describes these checks.

FIRMWARE STORAGE. The program memory is integrated into the microcontroller. Is the OEM testing sufficient or not sufficient? Do not jolly anyone along; doing so will eventually cause the project to fail. Describe the manufacturing tests here, including burn-in. Describe any integrity checks.

How many erase cycles is the non-volatile memory rated for? How often will the non-volatile memory be modified? What is the life span of the product?

15.7. TEMPERATURE OPERATING RANGE

The electrical design will operate over a temperature range sufficient to support design verification, manufacturing operations, and manufacturing test. State what these are. Will "industrial" rated parts be required? Do the temperature limits support the product spec?

15.8. EMI CONSIDERATIONS

Are the buck/boost regulators configured specifically? Are there IO pins to be held to a ground (or other) state? Chokes? Layout considerations? Cabling requirements?

15.9. BOARD LAYOUT

Are there elements to be aware in board layout? For example:

- RF antenna
- Crystal oscillators and parasitic capacitance
- High cycle current flow, around buck/boost voltage converters and current measurement devices
- High current flow, such as in recharge

16. POWER CHARACTERISTICS

Power usage / rating / characteristics are TBD.

How much is the power consumption.. at start-up? running? sleep?

Is there low power requirements?

16.1. TUNING SOFTWARE ACTIVITIES FOR LOW POWER

[describe]

What are the controls?

16.2. CONFIGURING THE MODULE FOR LOW POWER

[describe]

Which GPIOs should be set high? Low?

Which GPIOs should be inputs? Which should be turned into analog inputs?

What clocks should be turned off? Which should be left on?

etc.

16.3. TESTING THE POWER STATE OF THE MODULE

[describe]

It is possible to tell when the module is in a particular mode, such ass in run mode or deep sleep mode? Describe how to do so

17. TRIM & CALIBRATION

Calibration:

- Is the device calibrated per design? Is it calibrated per unit?
- How will the unit be trimmed/calibrated?
- Does the device self-calibrate?

Describe the calibrations that must be performed and how that is supported.

18. THE SAFETY & INTEGRITY MODEL

Describe how the electronics, microcontroller and its firmware control approach safety.

18.1. FEATURES THAT ARE CONTROLLED BY THE OPERATOR

Describe the features that are controlled by the operator and how they interact with these layers

18.2. FEATURES THAT MEASURE THE CURRENT STATE TO ENSURE THAT WE ARE WITHIN THE SAFE OPERATING REGION.

Describe the features that ensure the system is within a safe operating region. How is the current state measured? For instance, a typical microcontroller includes several features that may be relevant:

- Input voltage monitoring
- Power supervisor / brown-out detect function
- Communication with subsystems

18.3. SECONDARY CONTROLS

18.3.1 Power supervisor, brown-out detect

Does the microcontroller include an internal brownout detect? Describe it.

microcontroller power supervision

18.4. FEATURES THAT PREVENT PROBLEMS

Does the design include features that prevent problems? Some special microcontroller might include:

- PWM break function
- Peripheral lock bits

18.4.1 PWM Break function

Some microcontrollers – usually those targeting motor controlled – have an a signal input that disables the PWM. For instance, an "emergency stop" (ESTOP) signal may be connected to this input (and other mechanisms), to stop the impelling a motor, even if the software has malfunctioned.

PWM break input

18.4.2 Peripheral locks

Several microcontrollers include a feature that "locks" the peripheral registers from further change. Is this feature present? Describe it and its potential use. Typically the registers are read/write until lock activation. Once the lock bits have been set, the protected registers – including the lock bits – are read-only until the microcontroller is reset, or a special unlock sequence is used. What registers are protected – the lock bits themselves (if not, indicate this)? the PWM configuration? GPIO configuration?

peripheral lock bits

18.5. FEATURES THAT CHECK THE INTEGRITY

This is the section to describe special features of the microcontroller that support checking the integrity of the microcontroller, the program, and it's execution. For instance:

- SRAM parity check
- Clock failure detection
- Watchdog timers
- CRC peripheral for fast storage and communication integrity checking.

18.5.1 SRAM parity checks

SRAM parity check

Does the microcontroller include SRAM self-check features, such as parity checking? If so describe it's configuration, and use for safety and system integrity.

18.5.2 Clock failure detection

Clock failure detection

Does the microcontroller include mechanisms to detect clock failure? For instance, many STM32 microcontrollers include a feature called *clock security system*: In the event that the external crystal oscillator and the internal oscillator have a significant mismatch, the microcontroller will enter into an NMI fault state.

does the microcontroller include other separate or independent clock?. These can also be used to cross-check the main oscillator.

19. PIN MAPS

Summarize the connectors to the module/sub-system.

- Connector 1
- Connector 2
- …
- Manufacturing test connector
- Programming connector

19.1. CONNECTOR 1

Table 27: *XYZ connector*

Pin	Label	Description

19.2. ARM CORTEX DEBUG, ETM AND PROGRAMMING

The table below describes typical JTAG/Single-Wire Debug programming connector for ARM Cortex processors:

Table 28: Programming pin map

Pin#	Pin Name	Label	Description
		TCK/SWCLK	Debug-interface Serial Wire clock input and JTAG Test Clock
		TMS/SWDIO	Debug-interface Serial Wire data input / output and JTAG Test Mode Select.
		TDO/SWO	Debug-interface Serial Wire viewer Output.
		TDI	Debug-interface JTAG Test Data In
		RESET	Resets the microcontroller
		V_{DD}	Power
		GND	Ground reference

20. ELECTRONICS DESIGN ANALYSIS

Take the schematic and electronics design description and analyze these to:

- Identify the interfaces and inputs/outputs
- Identify the required delays between steps
- The max time between steps
- Design identifies more specific actions that can be taken
- The function role & groups: UART, I2C, SPI, PWM, ADC, DAC, memory interface, etc.
- Equations, system model
- Catalog of functions: role of the signals, parameters to gather
- Pin maps of programmable elements (microcontroller / FPGAs, etc). Sort by function Biasing. The speed / rate, and other parameters
- Pin maps to function
- Pin Biasing
- Speed / rate and other parameters
- Catalog of function: roles of signals, parameters to gather
- Identify time delays, max sampling rate, max sampling time, max time between steps
- Map hardware to function and signals. Design identifiers more specific actions that can be taken.
- Connector: ESD, reverse polarity, overvoltage; connector name, pin names; internal node names

20.1. SCHEMATIC REVIEW

Reviewing the schematic should look to identify:

- Are the related components of a sub-circuit located together?
- Is the flow following convention?

- Is the signal arrow following the nominal direction of signal flow?
- Does the design address the requirements?
- Are there requirement that the electronic design should address… but does not?
- Are there elements of the electronic design not adequately covered by the requirements? Should requirements be written to clarify commitments & testable behaviour?
- Are there missing requirements? Specifically, are there product standards requirements that are missing?

CHAPTER 6

Software Realization

This chapter describes how to convert a body of requirements (in conjunction with a hardware design description) into a software implementation

- Overview
- Initialization of the board and configuring the microcontroller function
- Declaring the variables – event flags and timers – needed for the requirements
- Signal processing – filters, PID control
- Implementing the requirements

21. OVERVIEW

The implementation approach includes support of testing:

- Testing against the requirements
- Testing the software
- Testing which elements of the requirements have been invoked.
- The steps of execution
- Special realizations

21.1. STEPS IN [TBD]

Steps in [TBD]

1. Initialize hardware. The in/outs, the pull-ups, the assignment to alternate function
2. Configuration tables of the hardware; connect hardware/framework function to pins /other
3. Configuration of the signal processing
4. Define event flags as conditions for the requirements. Define timers for conditions

22. BOARD CONFIGURATION / INITIALIZATION

Take from schematic pin map

Configure tables, # defines

Configure clocks

Configure peripheral rates

[more description]

23. PARAMETERIZED SIGNAL PROCESSING

This section discusses [intro]

- IIR Filters
- PID controls
- Hysteresis: conversion to digital
- Clamps / min / max
- Classification Tables
- Events

23.1. IIR FILTERS

Infinite-impulse response filters (IIRs) are an easy to implement form of filters – and usually take less CPU power than other realization methods. There is stock of techniques to convert filter specifications into an IIR implementation. The IIR filter can implement any filter specified as a transfer function in the form:

$$H[z] = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Equation 7: *Filter transfer function*

The 5 coefficients for 2nd order IIR filters are called a_0, a_1, a_2, b_1, and b_2. The code below summarizes the kernel of the filter computation:

$$out[n] = \begin{matrix} b_0 in[n] & + b_1\, in[n-1] + b_2\, in[n-2] + \cdots \\ & + a_1 out[n-1] + a_2 out[n-2] + \cdots \end{matrix}$$

Equation 8: *Recursive filter evaluation*

At most, the software need retain only the last 2 input values and the last 2 output values.

23.1.1 Converting filters to IIR filters

There are recipes to implement an IIR filter for each of the parameterized filter specifications. The filter type and its parameters (parameterized specification) identified during the analysis phase.

The 12 kinds of filter specifications given in chapter 3 can be converted into IIR coefficients for the transfer function given above. "Each of these is designed to optimize a different performance parameter. The complexity of each filter can be adjusted by selecting the number of poles and zeros,"

Table 29: *Filter design types*

Filter	Description
Bessel	Preserves the shape of the waveform. "The Bessel filter has no ripple in the passband, but the rolloff is far worse than the Butterworth."
Butterworth	"Optimized to provide the sharpest rolloff possible without allowing ripple in the passband. It is commonly called the maximally flat filter, and is identical to a Chebyshev designed for zero passband ripple."
Chebyshev	Fast roll-off (drop in amplitude), changes waveform shape. Bad for anti-alias filters. Good for removing separate frequencies.

Elliptic	"Allows ripple in both the passband and the stopband. Although harder to design, elliptic filters can achieve an even better tradeoff between roll-off and passband ripple."

It is recommended to limit IIR filters to 2nd order bi-quad filters. Filters with higher orders are rare in practice, and better accomplished by cascading bi-quads. "High-order IIR filters can be highly sensitive to quantization of their coefficients, and can easily become unstable." *Smith 1997*

23.1.2 Special filters and how to specify them

Below is a synopsis of the how to form common filters that would be used in this application. Note: The conversion to coefficients takes into account the sampling rate.

DC REMOVAL FILTER has

$b_0=1, b_1=-1, b_2=0, a_1=0 .. -0.95, a_2=0$

Equation 9: *DC removal filter*

EXPONENTIAL SMOOTHING has coefficients of the form:

$$b_0 = \alpha$$
$$b_1 = 0$$
$$a_1 = 1 - \alpha$$
$$\alpha = e^{-1/(Fsample\tau)}$$

Equation 10: *Exponential smoothing filter construction*

{all other coefficients are zero}

The process to form the coefficients for other filters based on their the type and kind of filter can be found in Smith (1997) and Redmon (2011)

23.2. PROPORTIONAL-INTEGRAL-DERIVATIVE (PID) LOOPS

Proportional-Integral-Derivative control loops are used when the reference signals imposed on the system are ramps or time-functions. A PID specification is given with the following parameters:

Table 30: *PID parameters*

Parameter	Description
K_D	The coefficient of the derivative term. By setting the K_d coefficient to zero get a PI. A PI control is used when the reference signals imposed on the system are steps or set-points.
K_I	The coefficient of the integral term
K_P	The coefficient of the proportional term
Out max	The maximum output value. The output and integral are kept ≤ to this value. This keeps the integral term from winding up.
Out min	The minimum output value. The output and integral are kept ≥ to this value. This keeps the integral term from winding up.

The PID specification can be converted to a form that uses the IIR module. This form has advantages over the conventional "naïve" implementation, including it is less glitchy.

Equation 11: *Mapping PID coefficients to IIR formulation*

$$b_0 = K_p + \tfrac{1}{2} K_i \Delta t + \frac{K_d}{\Delta t}$$

$$b_1 = -K_p + \tfrac{1}{2} K_i \Delta t + 2\frac{K_d}{\Delta t}$$

$$b_2 = \frac{K_d}{\Delta t}$$

$$a_1 = -1$$

$$a_2 = 0$$

Notes:

- Normalize the input/set point to 0..1 or -1..1
- Clamp the history value to prevent windup

23.2.1 Testing

The filter (and ADC and signal chain) can be tested by injecting a signal and checking the output:

6. Connect probe to input test points
7. Connect probe to output test points
8. Route an ADC input to the filter input in software
9. Route to filter output to DAC in software
10. Inject step pulse into the input
11. Watch the input test points, output test points and the DAC output for the step response.

23.3. HYSTERSIS

[control bands] [also, support the thresholds varying based upon operating conditions]

[describe]

23.4. CLASSIFICATION TABLES

This section describes the implementation of classification tables in software. Such a table maps input conditions to an output state:

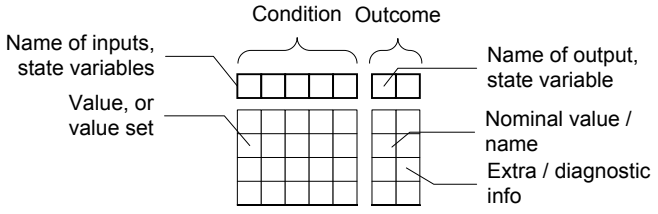

Figure 27: *Property lookup table*

The rows should not compete or conflict. (This is something that should have been looked for in the analysis phase).

I will look at two different ways of converting this decision table to code:

1. A simple, naïve if-then-else approach
2. An approach that maps the table to data structures, with the potential to reuse code

23.4.1 Mapping a decision table to if-then-else (approach #1)

The approach is three parts:

1. Each row maps to "if-then" structures
2. Between each row is an "else"
3. The last row is the *default value*

The template for the table is:

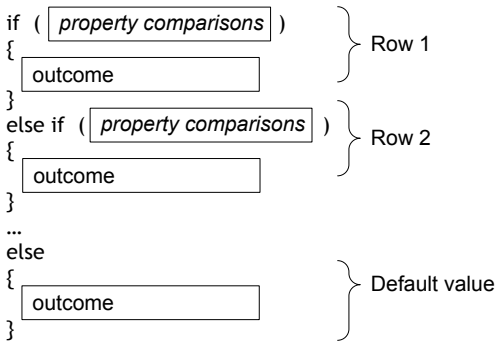

Figure 28: *How a decision table can be implemented*

The template for a single row is:

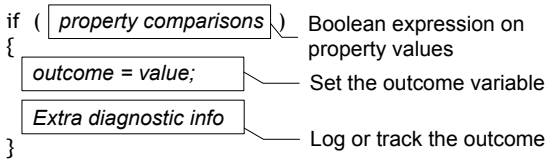

Figure 29: *How a decision table row can be implemented*

The column names can be event flags, or property names. This may include referring to property values at the previous time step. The process to create property comparisons is::

1. Skip the column if the variable is marked "do not care" (or any or blank)
2. If specific value or state comparison
3. If marked as becoming a specific value or state (i.e., was not in the previous time step)
4. All of the comparisons will be joined with an "&&"

The property comparison can be composed of current values, previous time step values… and event flags. A boolean of expression comparing composed f the following:

- Current property values (being compared against other property values or threshold)
- Property values at a previous time step
- Event flags (being evaluated as true, not true, or the same as other event flags)

23.4.2 Mapping a decision table to data structures (approach #2)

Mapping a decision table to data structures is a slightly more complex approach. [Is there any advantage? Can it do everything? I'm increasingly skeptical] One advantage is that it can be reprogrammable without requiring recompiling the firmware.

This form of classification table maps to the structures here:

- The table maps to a table structure.

- Each row maps to a row structure. The rows are stored as an array, referenced by the table structure.
- The cells in the row map to a pattern structure.

The software employs the following pattern to represent a classification tables:

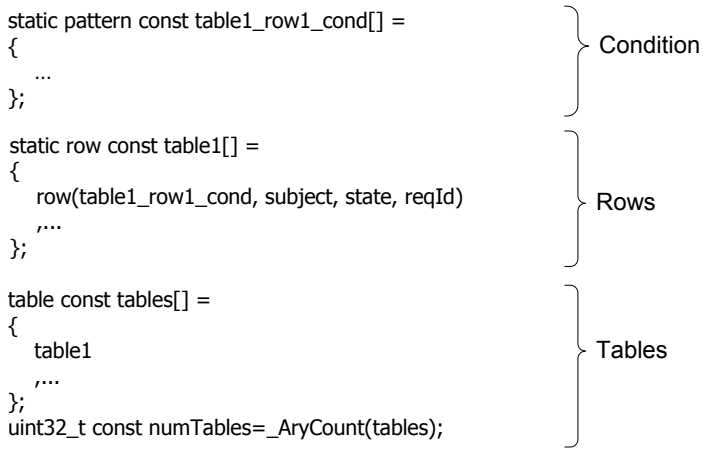

Figure 30: *Overview of the code for the classification table*

The process to convert the classification table into code is:

1. Create an array of `row` structures for the table (see the middle of the figure above).
2. Start with the first row. Create a `pattern` array of structures, as show above. Create a list of the conditions for that row. There is usually only one pattern entry for the conditions.
3. Add a reference to condition and output value to the `table` array of table rows
4. Repeat for the remaining rows
5. Add the table to the array of tables, `SMap_tables[]`

23.5. EVENTS

The *trigger* for an action is an event flag. [Todo naming convention for events]. An event flag is set if one of the following occurs:

- The hardware has signaled that an event has occurred.
- Properties match a value range or pattern
- Counts of events are within a range
- Message, or other external event has been received
- Timer expiration

event flags

EVENT FLAGS are variables that correspond to an event. This may be a timer expiring, an activity starting or completing, or observance of an external event – such as button press, or a value outside of operating limits.

The use of event flags may also serve in "white box" testing. The flag can be observed in the debuggers "live watch" window. The "live watch" can also inject the signal, to test the correct behaviour of triggered actions.

The template for detecting a property-value based event:

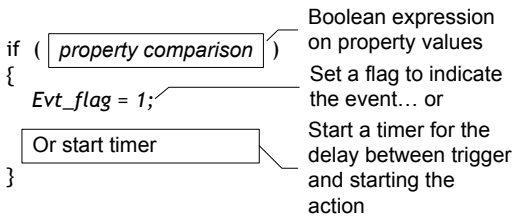

Figure 31: *Typical check of property values to indicate event*

The property comparison: current values, previous time step values… and event flags. A boolean of expression comparing composed of the following:

- Current property values (being compared against other property values or threshold)
- Property values at a previous time step
- Event flags (being evaluated as true, not true, or the same as other event flags)

23.5.1 Signal state and signal transition

signal state clauses

The clauses that are predicated on the state of an input signal can be translated to source code as:

```
(stateNow -> debouncedInputs & (mask of input signals)) == mask of states if each input signal
```

signal transition clauses

The clauses that are predicated on the state transition of an input signal to a target state can be translated to source code as:

```
    ((stateNow -> debouncedInputs & (mask of input signal)) == mask of target state)
&&  ((statePrev -> debouncedInputs & (mask of input signal)) != mask of target state)
```

23.5.2 Timers and event flags

By convention the name of event flags is predictable from the timer name. Where the timer is prefixed "Tmr_" followed by name, the event flag is "Evt_" prefix followed by the name.

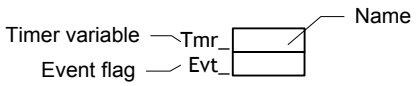

Figure 32: *Timer and event flag naming convention*

24. REALIZING REQUIREMENTS

intro

- Actions
- Triggers
- Wrap up, review of the software design & requirements

24.1. ACTIONS

The template for the flag triggering the start of the action:

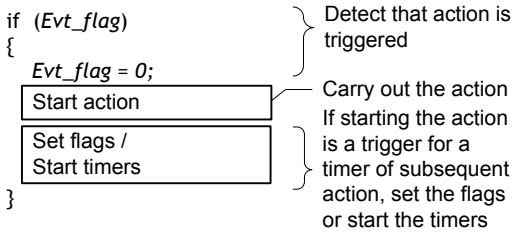

Figure 33: *Typical check to initiate the action*

pre-conditions – usually at the previous time step

24.2. OTHER PROPERTIES

Counting events / actions / etc

24.3. SOFWARE REVIEW

Reviewing the software should look to identify:

- Does the design address the requirements?
- Are there requirement that the software design should address… but does not?
- Are there elements of the software design not adequately covered by the requirements? Should requirements be written to clarify commitments & testable behaviour?

[This page is intentionally left blank for purposes of double-sided printing]

Appendices

This part provides supplemental material:

- ABBREVIATIONS, ACRONYMS, & GLOSSARY. This appendix provides a gloss of terms, abbreviations, and acronyms not defined in the other appendices.
- COMMON NOUNS. This appendix lists common nouns, including peripherals and sensors
- COMMON PROPERTIES. This appendix lists scientific units/dimensions.
- COMMON CATEGORICALS. This appendix lists the common categoricals.
- COMMON COMPARISON. This appendix lists the common comparables, comparatives, superlatives
- COMMON, WELL-KNOWN EVENTS. This appendix lists the common well-known events
- REQUIREMENTS MAP. This appendix maps each requirement to design elements that address it
- CODE COMPLETE SPECIFICATION REVIEW CHECKLISTS. This appendix reproduces checklists from *Code Complete, 2nd Ed* that are relevant to specification reviews.

[This page is intentionally left blank for purposes of double-sided printing]

APPENDIX A

Abbreviations, Acronyms, Glossary

Table 31: Common acronyms and abbreviations

Abbreviation / Acronym	Phrase
ADC	analog to digital converter
AIn	analog input
ANSI	American National Standards Institute
Calc	calibrate calibration
Conv	convert conversion
CRC	cyclic redundancy check
DIn	digital input
DIO	digital input/output
DOut	digital output
EEPROM	electrical-erasable programmable read-only memory
ESD	electro-static discharge
GPIO	general purpose IO
Hdw	hardware
JTAG	Joint Test Action Group
OPC	operator presence control
Pwr	power
Recv	receive
RPM	rotations per minute
SAR	successive-approximation converter (a type of ADC)
SDK	software development kit
Snd	send
STM32	A microcontroller family from ST Microelectronics
Sys	system
SWD	single wire debug

Tst	test
Tim	time
UART	universal asynchronous receiver/transmitter

Table 32: *Glossary of common terms and phrases*

Phrase	Description
abnormal condition	A condition when an operating variable has a value outside of its normal operating limits.[5] See also *fault, normal operating condition.*
acquisition time	see *sampling time.*
application logic	Application logic is a set of rules (implemented in software, or hardware) that are specific to the product.
black-box testing	Testing technique focusing on testing functional requirements (and other specifications) with no examination of the internal structure or workings of the item.
break function	The break function protects the power FETs driven by the PWM; when a break signal is received, the PWM outputs are disabled.
capacitive divider	
control function (class B)	Those "intended to prevent an unsafe state of the appliance. Failure of the control function will not lead directly to a hazardous situation" (EN 60730-1:2011 section H.27.1.2.2)
conversion time	The duration that it takes an ADC to sample and converter an input voltage into a measurement. This is the sum of the sampling time and the hold time.
data integrity	That the stored data – such as program memory – is intact, unchanged, in the expected order and complete; that is, that the entire program memory area matches *exactly* with the data defined for a particular revision.
data retention	The ability for a storage to hold bits
debounce	Switches and contacts tend to generated multiple rising & falling edges when coming into contact; debouncing removes the extra signals.
defect	A flaw in design or implementation esp. one that may lead to failure.
design document	A design document explains the design of a product, with a justification how it addresses safety and other concerns.
$failure_1$	A failure "is a permanent interruption of a system's ability to perform a required function under specified operating conditions." (Isermann & Ballé 1997).
$failure_2$	An incident or event where the product does not perform functions (esp. critical functions) within in specified limits.
fault	A component not meeting its specifications.
fault tolerant	"The capability of software to provide continued correct execution in the presence of a defined set of microelectronic hardware and software faults."[UL 1998]
flash	A type of persistent (non-volatile) storage media.
high-level document	System specification, customer inputs, marketing inputs, etc.

[5] Modified from http://www.wartsila.com/encyclopedia/term/abnormal-condition

hold time	The duration (after the sample capacitor has been charge) during which the voltage is converted into a measurement value. also called *settling time*
impedance divider	
integrity	"The degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data."[UL 1998]
initialization	Places each of the software and microcontroller elements into a known state; performed at startup.
integrity check	Checks to see that a storage unit has retained its data contents properly and that the contents have not changed unintentionally.
normal operating condition	A condition when each of the operating variables (flow, pressure, temperature, voltage, etc.) has a value within of its respective normal operating limits. See also *abnormal operating condition, fault.*
oversampling	
parity check	A simple form of error detection. Each byte in SRAM has an extra check bit that can catch memory errors.
power regulator	
power source	Where the electric energy comes from.
protective control	A control whose "operation … is intended to prevent a hazardous situation during abnormal operation of the equipment" [EN60730-1]
requirement	Defines what an item must do.
resistive divider	
sampling time	The duration that an ADC connects its sample capacitor to an input and allows it discharge/charge so that its voltage matches the input voltage. also called *acquisition time*
settling time	see *hold time*
single wire debug	An electrical debugging interface for the ARM Cortex microcontrollers.
τ_{ADC}	The time constant of the analog converter sampling capacitor, its impedance and the analog source impedance.
$\tau_{Divider}$	The time constant of resistive divider. This divider is often the input to an analog to digital converter.
test report	A document describing how a product performed under test.
white-box testing	Testing technique focusing on testing functional requirements (and other specifications), with an examination of the internal structure or workings of the item.

APPENDIX B

Common Nouns

This appendix lists the "well-known" common nouns. Note: these are distinguished from an abstract noun

Table 33: Common Noun acronyms and abbreviations

Abbreviation / Acronym	Phrase
ADC	analog to digital converter
Btn	button
Ch	channel
DAC	digital to analog converter
DMA	direct memory access
I^2C	inter-IC communication; a type of serial interface
IRQ	Interrupt request
ISR	Interrupt service routine
LED	Light emitting diode
MCU	microcontroller (unit)
Mem	memory
MPU	memory protection unit
NMI	non-maskable interrupt
NVIC	nested vector interrupt controller
PMAC	permanent magnet AC motor
PWM	pulse width modulator
RAM	random access memory; aka data memory
SMBus	system management bus
SRAM	static RAM
TMR	timer
UART	universal asynchronous receiver/transmitter
WDT	Watchdog timer

Table 34: *Common kinds*

Kind	Kind of	Description
analog to digital converter	peripheral	An analog to digital converter measures a voltage signal, producing a digital value.
connector		
cyclic redundancy check		A form of error-detecting code. A check value is computed from a block of data.
digital to analog converter		A digital to analog converter is used create a voltage signal from an internal value.
direct memory access	peripheral	A microcontroller peripheral that moves data to/from another peripheral from/to data memory; this is useful to reduce work done in software.
H-bridge		An electronic circuit that allows a voltage to be applied to a load in either direction.
Hall cell		A type of sealed switch the closes in the presence of a magnet; this is used to sense the position of the rotor.
microcontroller		
non-maskable interrupt	interrupt	A type of microcontroller fault.
non-volatile memory	storage	A storage mechanism that will preserve information without power.
peripheral lock		The microcontroller's peripheral registers can be locked, preventing modification until microcontroller reset.
single wire debug	interface	An electrical debugging interface for the ARM Cortex microcontrollers.
switch		
timer	counter	Increments (or decrements) regularly, setting a flag and (optionally) raising an interrupt when it expires.
watchdog timer	timer	A hardware timer that automatically resets the microcontroller if the software is unable to periodically service it.

Table 35: *Common sensors*

Phenomena	Sensor	Electrical Output
Accelerometer	Force balance	
	pendulum	
	Piezo-electric	
	Servo	
Acoustic		
Chemical	CO Sensor	Voltage, charge
	Ion Sensor	Current
	pH Electrode	Voltage
	Photodiode (turbidity, colorimeter)	Current
	Solution Conductivity	Resistance/current
Flow	Hot-wire Anemometer	Resistance
	Magnetic Flow-meter	Voltage

	Mass Flow-meter	Resistance, Voltage
	Mechanical Transducer (turbine)	Voltage
	Ultrasound/Doppler	Frequency
Fluid Level and Volume	Capacitor	Capacitance
	Mechanical Transducer	Resistance/Voltage
	Switch	on/off
	Thermal	Voltage
	Ultrasound	Time Delay
Force, Weight, Torque, Pressure	Load Cell	Resistance
	Mechanical Transducer	Resistance, Voltage, Capacitance
	Piezoelectric	Voltage, or charge
Humidity	Capacitive	Capacitance
	Infrared	Current
Ionic concentration	pH Probes	
	dissolved oxygen	
Light	Photodiode	Current
Magnetic	Hall Effect	Voltage
	Magneto-Resistive	Resistance
Motion and Vibration	Accelerometer	Voltage
	LVDT	AC Voltage
	Microphone	Voltage
	Piezoelectric	Voltage or Charge
	Ultrasonic	Resistance, Voltage, Current
Position	interferometer	
	potentiometer	
	LVDT	
	RADAR & ultrasound	
	rotory variable-reluctance differential transform (RVDT)	
	shaft encoder	
	star tracker	
Pressure	force-displacement	
	strain-gauge	
Proximity	Capacitance	Voltage, Frequency
	Inductance	Current, Frequency
	Resistance	Voltage, Current
Radiation	Geiger counter	
Strain	Piezo-electric	voltage
	Piezo-resistive	Resistance
Temperature	bimetallic	
	IC	Voltage
	RTD	Resistance
	Thermocouple	Voltage
	Thermopile	Voltage
	Thermistor	Resistance
	Infrared	Current
Touch	Capacitance	Voltage
	Inductance	Current

	Resistance	Frequency
Velocity	Doppler effect transducers (radar, ultrasonic)	
	Rate gyroscopes	
	Tachogenerators	

APPENDIX C

Properties

Abbreviation / Acronym	Phrase
AVG	average
Hz	Hertz; 1 cycle/second
Idx	index
Len	length
Num	number
On	on
Off	off
Sec	second(s)

Table 36: *Common acronyms and abbreviations*

Property	Kind of	Dimension	Possible values
brightness	number	lumens	
gender			{male, female, neuter}

Table 37: *Glossary of common properties*

It is suggested (strongly) that the following units be employed and used consistently for their dimension:

String	Description
A	Amps
°C	Degrees Celsius
C	Coulombs
Hz	Frequency, or cycles/second
J	Joules
V	Volts or voltage

Table 38: *Common unit*

APPENDIX D

Common Categorical Values & States

This appendix describes common categoricals.

Table 39: Glossary of common categoricals

Phrase	Description
female	
male	
on	
off	

Colors – such as red green, blue – are often treated as categoricals. Even if they can be arranged as a structure of ordinal elements.

APPENDIX E

Common Comparison

Integers and reals: ==, ≤, <, >, ≥

For floating point, the following transforms do not apply

A is not equivalent to

a == b b == a

a < b b > a

a ≤ b b ≥ a

a ≥ b b ≤ a

Both relative comparisons are false is either a or b is a NAN.

Adjective	Property	Comparison
brighter	brightness	>

Table 40: *Common comparative definitions*

Adjective	Property	Comparison	Auxiliary
brighter	brightness	>	
bright	brightness	>	TBD

Table 41: *Common comparable definitions*

The comparison is grouped by the property dimension below:

dimension	Adjective	Property	Comparison	Auxiliary
length	*longer*	length	>	
	long	length	>	TBD
duration	*longer*	duration	>	
	long	duration	>	TBD

Table 42: *Comparable definitions for an instance*

APPENDIX F

Common Events

This appendix describes common events, excluding those derived from an action beginning or ending.

Table 43: Common acronyms and abbreviations

Abbreviation / Acronym	Phrase
IRQ	Interrupt request
POR	Power on reset

Table 44: Glossary of common events

Phrase	Description
error	An error is the occurrence of an incorrect (or undesired) result.
exception	A special condition – often an error – that changes the normal control flow. On an ARM Cortex, this can cause the processor to suspend the currently executing instruction stream and execute a specific exception handler or interrupt service routine.
fails	An incident or event where the product does not perform functions (esp. critical functions) within in specified limits.
$fault_1$	A fault is an abnormal condition, or other unacceptable state of some subsystem (or component) that will disallow operation. See also *abnormal condition, normal operating condition.*
$fault_2$	A fault is represented an interrupt or exception on ARM processors that pass control to handler of such an abnormal condition.
hard fault	A type of microcontroller fault.
power off	When energy is no longer (or is to no longer be) applied to the system or subsystem.
power on	when energy is to be applied to the system or subsystem
power on reset	A type of microcontroller reset that occurs when power is applied to the microcontroller; release from reset allows software to execute.
single event upset	An ionizing particle flipped a bit or transistor state
start up	
system tick	The (regular) expiration of a key "system" timer.
watchdog reset	A microcontroller reset triggered by the expiration of a watchdog timer.

APPENDIX G

Code Complete Specification Review Checklists

Adapted from **STEVEN C. MCCONNELL,** *CODE COMPLETE, 2ND ED.*

Source: https://github.com/janosgyerik/software-construction-notes/tree/master/checklists-all

25. CHECKLIST: REQUIREMENTS

25.1. SPECIFIC FUNCTIONAL REQUIREMENTS

- ☐ Are all the inputs to the system specified, including their source, accuracy, range of values, and frequency?
- ☐ Are all the outputs from the system specified, including their destination, accuracy, range of values, frequency, and format?
- ☐ Are all output formats specified for web pages, reports, and so on?
- ☐ Are all the external hardware and software interfaces specified?
- ☐ Are all the external communication interfaces specified, including handshaking, error-checking, and communication protocols?
- ☐ Are all the tasks the user wants to perform specified?
- ☐ Is the data used in each task and the data resulting from each task specified?

25.2. SPECIFIC NON-FUNCTIONAL (QUALITY) REQUIREMENTS

- ☐ Is the expected response time, from the user's point of view, specified for all necessary operations?
- ☐ Are other timing considerations specified, such as processing time, data-transfer rate, and system throughput?

- ☐ Is the level of security specified?
- ☐ Is the reliability specified, including the consequences of software failure, the vital information that needs to be protected from failure, and the strategy for error detection and recovery?
- ☐ Is maximum memory specified?
- ☐ Is the maximum storage specified?
- ☐ Is the maintainability of the system specified, including its ability to adapt to changes in specific functionality, changes in the operating environment, and changes in its interfaces with other software?
- ☐ Is the definition of success included? Of failure?

25.3. REQUIREMENTS QUALITY

- ☐ Are the requirements written in the user's language? Do the users think so?
- ☐ Does each requirement avoid conflicts with other requirements?
- ☐ Are acceptable trade-offs between competing attributes specified—for example, between robustness and correctness?
- ☐ Do the requirements avoid specifying the design?
- ☐ Are the requirements at a fairly consistent level of detail? Should any requirement be specified in more detail? Should any requirement be specified in less detail?
- ☐ Are the requirements clear enough to be turned over to an independent group for construction and still be understood?
- ☐ Is each item relevant to the problem and its solution? Can each item be traced to its origin in the problem environment?
- ☐ Is each requirement testable? Will it be possible for independent testing to determine whether each requirement has been satisfied?
- ☐ Are all possible changes to the requirements specified, including the likelihood of each change?

25.4. REQUIREMENTS COMPLETENESS

- ☐ Where information isn't available before development begins, are the areas of incompleteness specified?
- ☐ Are the requirements complete in the sense that if the product satisfies every requirement, it will be acceptable?
- ☐ Are you comfortable with all the requirements? Have you eliminated requirements that are impossible to implement and included just to appease your customer or your boss?