**Exploring Your System Architecture**

through scripting how your system may be intending to operate

**Cornell Cup USA – Arm Enabled**

**Prof. David R. Schneider, Cornell Director of MEng Studies in Systems Engineering, drs44@cornell.edu**

You and your team have an idea for the most fantastic project any of you have thought of! Each of you has a different set of skills essential for making this project a reality and so each of you set out on developing your own part. It isn’t long however until your team begins to have conversations like the one just below. This team is working on a health monitoring wearable device that predicts a heart attack before it’s likely to occur. The device is thought to consist of both a chest strap with sensors coming off it to be worn under a shirt, and a wrist band device with its own sensors and a smart watch-like user interface. The team of 4 has broken themselves into roles on Communications between device components, Algorithms to interpret sensor data, Sensors to collect & filter the sensor data, and Mechanical to building the housing for the electronics and wearable components.

Communications teammate: “Okay great I think I’ve got the communications protocols all set up for the chest and wrist parts to talk wirelessly to each other”

Algorithm teammate: “Great, make sure you send it to me at 60Hz”

Communications teammate: “60Hz?!”

Algorithm teammate: “Or faster…”

Communications teammate: “The comms can only run at 30Hz”

Sensors teammate: “30 Hz is actually way faster than it needs to be. This sensor only update twice a second, so…”

Mechanical teammate: “Wait let me take a look at that sensor. I thought you said it just needed to be attached somewhere in the front. The housing I built doesn’t have any room for all those wires and the fat connector”

Sensors teammate: “That connector is standard. What were you expecting?”

Algorithms teammate: “Wait back to the data rate. If the sensor can’t provide 60Hz, there’s no way this algorithm is going to provide any valid data. It’s just going to spit out garbage.”

Communications teammate: “Well it’s going to be 30 Hz of garbage ‘cuz that’s all the comms can handle”

Mechanical teammate: “Well it’s not even going to be that because there ain’t no way we can fit that sensor into our ergonomic design. Your sensor’s not even good enough for the algorithm, so you’ll have to change the sensor”

Sensors teammate: “No you’ll have to change your housing, and that algorithm is way too demanding.”

Communications teammate: “Well I guess the housing also depends upon how much room we have for the batteries”

Sensors teammate: “Well I’m just handling the power requirements for the sensors so I can tell you what they need. Who’s taking care of power overall?... Nobody??? Comms, I thought you were handling all of the things between the devices?!”

Communications teammate: “No I’m just doing the comms. That’s it.”

Algorithms teammate: “You guys must be joking. I’m doing all the hard work making this algorithm run and you guys can’t even hook up a few sensors?!”

Mechanical teammate: “Actually we’re all screwed. We forgot to check how the sensor will work if there’s any moisture, like sweat. Which means we need a different sensor, which means I need to redo the housing, and I think that also means your algorithm won’t get the data it needs. But at least it will be faster for us to do it all again the second time, right?”

Communications teammate: “Well at least the comms still works.”

The conversation breaks down to being less than productive at that point… A lot of potentially good work was done, perhaps all working fine on its own, but when they went to put their parts together, that’s when issues began to be noticed. What the team needed was to have some way to outline what are all of the key functions that the system had to do and help highlight how they needed to interact with one another. What they needed is referred to as systems architecture.

**Overview**

This guide introduces the Functional Flow Block Diagram (FFBD) as a great introductory systems architecture tool that is commonly used in a wide variety of areas from large industries and government agencies to smaller entrepreneurial and academic efforts. It’s possibly one of the simplest systems architecture tools out there. It’s also quite flexible and hence is a great one to start a project with while you’re still figuring out your system architecture. Overall it can be very helpful to get all of your team onboard with a general plan so they at least recognize how their different parts might be related. This in turn can help spur them on to having more in-depth conversations, recognizing the complexity of everyone’s tasks (even if they may not understand all of the technology behind it) and keep each other “in the loop” about their progress and most importantly any interfaces between each other’s work.

If you’re familiar with Activity Diagrams or IDEF0 diagrams you can use one of those for similar purposes as well. Similarly, there are other software tools like Visio and CORE that can help to create diagrams like these but we’ll be using PowerPoint in our example as it is relatively easily available and easy to use. Regardless of what tool you use though, Expect to Iterate! And Iterate! Unless your system is remarkably simple (like “alarm clock” simple) defining a systems’s architecture is considered a discovery process. As you strive to describe your system, you begin to learn more about the intricacies of your system, which in turn may inspire you to change your system for the better. Rarely are our first ideas the best. By exploring them, we can almost always discover new important aspects and needs that must be addressed, and hence way and better ways to develop our solution.

The FFBD has been used in some form for possibly over a hundred years, and the fact that it is still used to today speaks to how useful it can be. Because it’s been around for so long though, there are several variations that exist, each with slightly different formatting rules and symbolic representations. In this guide we will describe a variation that isn’t hard to adjust to many other variations, and we provide some common practices from other variations that you may find helpful. The most important thing though is that whatever variation you use, your team is consistent in the way they use it to help ensure it’s properly interpreted by all. (Think of all the issues that have arisen from mismatched measurement units – any kind of inconsistent communication can lead to all kinds of unforeseen problems).

Other kinds of system architecture tools do exist as well that can meet many of the same design needs as an FFBD, such as Activity Diagrams or IDEF0. However, it is the flexibility and ease of use of the FFBD and its variations that continues to make it a great tool used widely in many professional circles.

**FFBD Example Walkthrough**

As the FFBD is a fairly straightforward tool, to introduce the FFBD it may be easiest to walkthrough an example so you can see how the FFBD is used to describe a system. We’ll define the components of the FFBD more formally below but hopefully this will help give some initial concept as to how the components can be used.

The example FFBD is created to describe the health monitoring wearable device that we started this guide with. But let’s first offer a little more of a description that on how the team might have described the project when they first got started: The wrist band should communicate wirelessly to the chest strap and each have their own battery and microcontroller to operate the sensors. The sensors monitor a variety of health statistics including blood pressure, heart rate, blood dissolved oxygen, and blood flow rate. Each sensor also has its own special requirements. For example, some need the wrist band and chest straps sensors to work together and others can potentially cause interference for one another. The system provides regular updates to the user but requires some information from the user as to their current activity (are they at rest, exercising, doing something stressful, etc.). If the system detects a potential issue, it notifies the user. Then depending upon the potential severity, it alerts both people in the local vicinity as well as emergency medical professionals.

Most of the paragraph above is captured in the first FFBD slide. In just this first FFBD slide though we can begin to see a little more detail. In general the FFBD is read from left to right and steps through the functions that must occur in its operation – essentially the FFBD is system’s operational flow chart. Separate, even many FFBDs could be created to describe a single system’s various modes of operation, use cases, even failure detection warnings, and emergency responses. As is done for most systems, at least initially, this example FFBD focuses on the overall operation of the system. But as we’ll see later, even this initial FFBD can be expanded in more detail wherever you want.

The example FFBD begins with indicating one of the first functions to perform in the system’s operation is to keep the device attached. To someone, like the person working on the sensors, this functionality may seem implicit. However, to the person working on the mechanical housing and ergonomic use of the system, this is a very critical function that needs to be highlighted. Similarly the mechanical housing person may think the next function of turning on the device as being implicit. However, this may aid the person working on power to help others recognize that the start-up of the system can have an initial transient.

In this simple example, recognizing these points may not have much an influence on how each person performs their separate work, but it can help start a discussion where teammates begin to realize more of what goes into each part of the system. The next block “Calibrate Device” however may inspire more conversation that will help to inform the mechanical housing person how to better design the housing to allow for easier, more reliable calibration.

From here the FFBD shows three paths of functionality that the systems performs in parallel. The top path shows that the system waits for user input and then either changes a setting or triggers a medical alert response. The middle row shows that the system is monitoring the person’s health, the results of which are analyzed via a standard algorithm and if necessary double checked with a secondary algorithm, perhaps to eliminate false positives or ensure possible issues aren’t missed. The results are then used to complete a report and trigger a medical alert response if necessary. In the third path, the system is performing some essential self-diagnostics and power monitoring. All three paths converge in displaying an update to the user and making any local announcements, which could be as simple as a loud buzzer alarm if warranted.

Additionally, we can see that a number of the arrows connecting the functional blocks have text labels. In some cases the labels indicate what information, or what kind of information, is being transferred. Other labels show the ending condition for an iteration loop. Still others specify additional requirements, such as the target rate a loop is occurring, i.e. the inner algorithm loop is specified as occurring at 60 Hz. Furthermore, as the FFBD may still be early in the design, one label is given a “?” as a means of indicating that there is still significant uncertainty here to be worked out.

The details on how to format the FFBD properly are covered in the Tools & Tips section, but even without going into that, this example demonstrates how the main functionality and operation of a system can be easily laid out for everyone to get a basic understanding – helping them to begin to recognize where interfaces between their individual work must occur.

**FFBD First Thoughts**

In reviewing this FFBD though, you may have had a couple questions arise, such as “Well this is a nice overview but how do I show everything that might be going on inside one of these functional blocks? – It could be rather complex and make the whole FFBD rather large and messy.” In some cases, a single block may represent an extensive set of operations itself. To delve deeper, a separate FFBD diagram can be created for what is occurring in any single block. For example, the “Monitor Health Functions” block “Monitor Health Functions” is described in more detail in slide 4 as if the “Monitor Health Functions” block where the opened up and we looked inside. This is referred to as hierarchical FFBD construction-- where an FFBD is actually comprised of many FFBD diagrams providing various levels of informational detail.

This can be a nice way to show various details when needed, or hide complexity that might not be necessary for everyone to see at first. But how did the people who created this example FFBD know to separate the functionality into these different blocks? How did they know what should be at one level of the hierarchy versus another? And how did they know how to organize the blocks so it would all “look nice” when they were finished? The honest answer is, they didn’t.

And they didn’t get it to be a way they were happy with on their first try. As mentioned above, the means of creating a valuable FFBD is through *iteration*. A couple rules of thumb that can help though is to always keep in mind that what is happening within each block at least represents a key step or function in the system’s operation. As one part of the FFBD may become more complicated, perhaps that part can be summarized as one or two functional blocks shown at a higher hierarchical level and then the rest of the complexity moved to its own separate FFBD diagram.

In many cases, what functions are grouped can also depend upon the skill sets of the people working on the project – i.e. the blocks tend to be split into functions that one or two key people (or groups of similarly skilled people) can take primary ownership of. As you create your FFBD you may likely recognize additional tasks that may need to be taken care of, which will help you recognize new functional blocks that should be added -- and adding in new functional blocks will help you recognize new tasks and interfaces. You’ll likely move certain functions up and down in the hierarchy, change the order in which operations were performed, even recognize opportunities for functionality that you hadn’t considered before. Iteration is the key to success. The more your team iterates and works together, the more they also learn about each’s needs. You will develop a better system and develop the system better together.

**FFBD Components, FAQ, & Tips**

Some design tools can be implemented in a step by step fashion. Other ones, like the FFBD, give you a few key components that you utilize towards your design. Below is a list that introduces the few key FFBD components. Each component is described in detail below and refers to the heart attack prediction device example to demonstrate their use:

* **Function Blocks**
* **Arrows: Triggers and Precedes**
* **Control Pairs (aka Summary Gates): IT, AND, and OR**
* **Titles**
* **Reference Blocks**
* **Data Information Blocks**
* **Arrow Shortcuts (aka Edge Shortcuts)**

After the FFBD components are described, below them are additional Tips and FAQs on the following topics. Anyone can create an FFBD fairly easily, but like many things the FFBD can be done well and be useful, or be done poorly by just going through the motions of putting blocks together without much thought. Hopefully by organizing the component descriptions and the tips/FAQ this way, you can refer to this guide as you need to build your own effective FFBD. But don’t worry if your first attempts don’t live up to your expectations. Even the simplest techniques in any sport, musical pursuit, dance, cooking exercise, etc. take time to perfect. The more you use this tool, the more benefit you we see from doing so.

* **What do I put in the functional blocks?**
  + *See Functional Blocks: Purpose*
* **Is it okay to cross arrows?**
  + *See Arrows: Formatting and Arrow Shortcuts*
* **My FFBD is really big. How can I make it easier to read?**
  + *See Functional Block: Formatting & Reference Blocks*
* **How should I decide to split up my FFBD?**
  + *See Reference Blocks*
* **How can I represent interactions with things outside of my system?**
  + *See Data Information Blocks*
* **Is there a right way to represent my system in an FFBD?**
  + *See FAQ & Tips #1*
* **Can I repeat functional blocks?**
  + *See FAQ & Tips #2*
* **Does my FFBD have enough clarity?**
  + *See FAQ & Tips #3*
* **Formatting is taking me a long time and I keep on having to clean up my arrows in PowerPoint. Is there a recommended way to do this faster?**
  + *See FAQ & Tips #4*
* **Can the FFBD help me determine what tasks to work on first? What if I’m uncertain about parts of the FFBD?**
  + *See FAQ & Tips #5*
* **Can the FFBD be used to represent a structural solution system architecture as well as a functional one?**
  + *See FAQ & Tips #6*

**Function Blocks**

Purpose: The functional block is the most essential FFBD building block. As mentioned above they can represent a single key function or operation or be representative of a much larger set of functions or operations, making them and the FFBD very flexible and versatile. As the name implies, the goal is for the text in the block to focus more on the functionality being performed versus the structural implementation that is doing the task. This can help people to better understand the operations as well as not feel locked into a particular structural implementation.

For example, imagine you needed to cool a component. The first solution might be to add a fan. However, later on you might recognize that a liquid cooling solution is better, or that a simple heat sink may work better, or you may simply be able to run the device intermittently, or that you could redesign the device’s housing to improve passive ventilation. Even in this very simple example, if you had listed a fan as the solution instead of “cooling” as a function, you and your teammates might assume that the solution must be a fan and might not have even considered other options when trying to implement your FFBD. Remember the mantra “Think Functionally, Not Structurally”.

Formatting: Their traditional shape is a square cornered box. They do not need to be all of the same size but there is a general goal to have only a small number of different sized boxes in the same FFBD. The function box is comprised of two parts, a larger main box on the bottom where the function text is written and a small “header” box on top that contains the function block ID.

Function block IDs (aka function reference numbers or function numbers) begin with a capitol “F” and then a period with a number after the period, or a series of periods and numbers, i.e “F.7” or “F.7.3” or “F.7.3.29”. The series of periods and numbers indicates the block’s position in the FFBD hierarchy. If the functional block is in a lower level FFBD of a hierarchy, another number is included to overall indicate where the functional blocks FFBD came from. For example, if on the top level FFBD there was a block with an ID of F.2 and then another FFBD was made to show what was “inside” the F.2 functional block, all of the function blocks in the lower level FFBD would have IDs of the form F.2.#. This can be repeated for as many levels of the hierarchy as needed, i.e. an FFBD 4 layers down in the hierarchy would have functional blocks with IDs of the general form F.#.#.#.# . An example of this is shown where the F.2.5 “Measure Health Conditions” block of slide \_\_\_\_\_ is broken down into more detail in slide \_\_\_\_\_.

Function block IDs are also unique, but the entire block can be repeated as described Tips & FAQs. In general, function block IDs are assigned in a left to right, top to bottom fashion within an FFBD. This isn’t required but makes it easier to read.

**Arrows: Triggers and Precedes (i.e. Solid and Dashed)**

Purpose: There are two main kinds of arrows, also known as “edges”, used in FFBDs. Trigger (solid line) arrows are the most common and state that the functional block at the origin of the arrow “triggers”, or is immediately followed by, the functionality of the function block at the end of the arrow. Precedes (dashed line) arrows indicate the functional block at the origin of the arrow will occur at some point before the function at the end of the arrow, where that amount of time may be considerable.

Arrows can be labeled with a variety of information. Different variations of the FFBD can have strict rules as to what labels are allowed but common labels can include:

* Information that is being passed between functions
* the expected time between functional blocks
* specifications on how one functional block influences another
* inputs and outputs to a functional block
* additional constraints or accuracy/variation information on any of the above

For arrows associated with AND, OR, or IT summary gates (described below), arrows can also commonly be labeled with

* conditions that specify why one operational path should be followed or another; common for OR gates
* the ending conditions for an iteration loop; common for IT gates
* variations between parallel paths, such as the amount of resources sent to each branch of an AND summary gate

There is a third kind of optional arrow used in what is referred to as an Extended FFBD (EFFBD) variation, which is discussed in the Data Information Blocks component description below.

Formatting: Arrows are drawn as rectilinear arrows with a filled in arrow head. The thickness of the arrows is typically the same thickness of the functional box boarders or “one setting” thicker. Unnecessary kinks or non-rectilinear arrows are consider very sloppy and should be consider a significant faux pas. The ends of arrows should be in direct contact with the functional blocks; spacing between arrow ends and functional blocks is often considered sloppy and possibly unacceptably unprofessional. Arrows should be made long enough to clearly distinguish whether it is a precedes or triggers arrow.

The text labels on the arrows should be placed typically above the arrow, but this is not always required, and this guideline may be broken for the purpose of improved readability. Most importantly, the text should be at a location that makes it obvious that it is associated with only one arrow. Text labels are commonly italicized and may be in the same or one font size smaller than the functional block text. If you want to have more than one text label on the same arrow, each text label is typically separated by a comma, although semi-colons are commonly used for separation as well.

Arrows should also not overlap other arrows or FFBD components. Some FFBD formats will allow arrow “jumps”, i.e. where one of the two intersecting arrows has a half-circles bump at the arrow intersection. In general, arrow overlaps should try to be avoided through the arrangement of the functional blocks. When significant arrow overlaps might seem necessary, an arrow shortcut component is often preferred to be used instead, as is described below.

**Arrow (aka Edge) Shortcut**

Purpose: In general, arrow overlaps should try to be avoided through the arrangement of the functional blocks. When arrow overlaps might seem necessary, an arrow shortcut component is used instead. An example of an arrow shortcut is shown in slide \_\_\_\_\_ coming out of the “Estimate Sweating” functional block. In this case we see that an arrow is coming out of the “Estimate Sweating” functional block goes into the “Set Microcontroller Settings” block, but the arrow is also split and goes into the circle “A” symbol.

Later we see there are two other circle “A” symbols; once which has an arrow going into the “Measure Heart Rate” block and one in the “Measure Dissolved Oxygen” block. The circle “A” symbols are referred to as the “A” arrow shortcut and together help show that in this FFBD, the output of the “Estimate Sweating” functional block goes into the “Set Microcontroller Settings” block, the “Measure Heart Rate” block, and the “Measure Dissolved Oxygen” block.

Formatting: arrow shortcuts should be used sparingly whenever possible but sometimes are necessary. They should only be used to connect blocks within the same FFBD diagram. Reference blocks, described below, should be used to connect functional blocks that are on different FFBD diagrams.

Please note that the initial split in the arrow coming out of the “Estimate Sweating” functional block and into the “Set Microcontroller Settings” block and the A arrow shortcut is not necessary. An arrow coming out of a functional block directly into an arrow shortcut is perfectly acceptable as well.

There can be several arrow shortcuts in the same FFBD, however they should each be given their own letter. Typically the first set of shortcuts is named “A”, the second set is named “B” and so forth. Like summary gates, the text inside the gate circle should be all caps, bold, and commonly within one font size of the functional block text, but always the same font size across all arrow shortcuts.

**Control Pairs (aka “Control Triggers”, “Summary Gates”, “Logic Gates” or “Gates”): IT, AND, and OR**

Purpose: Control pairs, also known as “summary gates”, or simply “gates” help give more information about the operational flow of your system. The 3 most common ones are called IT, AND, and OR which respectively indicate that part of the system function is done:

* repeatedly, (IT)
* happens in parallel, (AND)
* or takes one operational path over other possible ones. (OR)

IT gates commonly have an arrow with a text label indicating the ending condition for the iteration. OR gates typically have a text label with the condition that an arrow / operational path, should be followed. If only one operational path coming out of an OR gate does not have label, that path is considered the default or standard operational path. AND gate arrows can sometimes be labeled as well, sometimes to indicate how much of a certain resource is split amongst the various parallel operational paths, but quite commonly AND gate arrows do not have any label.

Formatting: Gates are always shown in a circle, and whenever possible, are used like curly brackets are used in many programming languages, i.e. they come in pairs that bookend the start and end of a section of the operational flow. (Having a matching ending gate isn’t always necessary in some FFBD formats.) Gates should always be the same size throughout the FFBD. The text inside the gate circle should be all caps, bold, and commonly within one font size of the functional block text, but always with the same font size used across all summary gates.

**Titles**

Purpose: Not so much a component to build an FFBD but a key organizational feature, every FFBD should have a title and typically a border as well. Like functional blocks, every FFBD diagram should have a unique title and ID, so that they can be referred to by other FFBD diagrams in a set of FFBDs.

Formatting: The title is located in the upper left corner. It is usually the same or one font size larger than the functional block text and is commonly bold. Each functional block in a FFBD diagram should a unique ID starting with the function number of that FFBD’s title. So all the function IDs in the FFBD titled “Function 1: Systems Initialization”, start with F.1. And all the function IDs in the FFBD titled “Function 2: Systems Operation”, start with F.2, etc.

In hierarchical FFBD diagrams, the title to your lower level FFBD diagram should be the function ID and name of the function block you are diving into. So for the example FFBD showing what’s inside functional block F.2.5 Measuring Health Conditions, we make “Function 2.5: Measuring Health Conditions” the title. Similarly, all functional blocks in this FFBD diagram are numbered with IS’s starting with F.2.5 starting with F.2.5.1, F.2.5.2, etc.

**Reference Block**

Purpose: Many times an FFBD becomes too large to fit on a single page, or is otherwise difficult to view all at once. Creating a hierarchy of FFBDs can be one way to address this but another way to have several FFBDs reference each other, even at the same hierarchical level, is to use reference blocks. Reference blocks are similar to arrow shortcuts but where arrow shortcuts are used to connect blocks in the same single FFBD diagram, reference blocks are used to connect blocks in different FFBD diagrams.

More than just providing a connection, reference blocks are commonly used to essentially section off significant parts of what would be a larger FFBD, to make a several more manageable FFBD diagrams. Sometimes this is done purely for legibility but the choice of where to section things off though should consider trying to create meaningful groupings of functionality such as

* Initialization and Shutdown
* Maintenance operations
* Handling of an error, emergency, or other special use case

Reference blocks tend to be used for where operational flow may branch off in a significantly different direction. An example of the use of reference blocks is where the FFBD of slide \_\_\_\_\_\_\_ is split after the “Calibrate Device” function block into the 2 FFBDs on slides \_\_\_\_\_ and \_\_\_\_\_\_\_. This allows the initialization part of the system to be separated from the main operation iteration loop and shutdown parts of the system.

Although this makes for a small FFBD diagram on initialization, small FFBD diagrams are perfectly fine to have. A good (even better) reason to split an FFBD apart other than size, is to make split FFBDs into parts that cover distinct sets of functionality or parts of the system operation, such as separating the initialization from the main operation parts. Similarly, the shutdown part of the FFBD could be separated into a separate FFBD diagram as well. However, perhaps since it’s even smaller and simpler than the initialization, the authors of this FFBD decided to keep it as a part of the larger one. Either way, would be fine to do.

Reference blocks are also used in slide \_\_\_\_\_ where parts of the higher level FFBDs are shown to tie into this lower level FFBD, such as the tie in to the “Power Sensors” and “Power Motor” blocks. This particular use of reference blocks also demonstrates how “what to show on one FFBD versus another” can be used to help signify functionality responsibility – in this case “Power Sensors” and “Power Motor” might pertain to just the connection of these devices into the main power system and this connection is something that the team decided is the responsibility of the Sensors teammate, leaving the overall power system to someone else.

Formatting: Reference blocks look like a large pair of square brackets that together are roughly the same size as your standard functional blocks. They do not have their own unique ID, but rather start with the ID associated with the functional block or other FFBD they are referring to. They then add the abbreviation “Ref” at the end of the ID to signify the reference further. The title of the FFBD or text of the functional block that this reference block is referring to is often included after the “Ref” as well.

**Data Information Blocks:**

Purpose: Data information blocks are sometimes called “data bubbles” as they are traditionally ovals or rounded corner boxes as shown in slide \_\_\_\_\_\_\_\_. Data bubbles are additional blocks added to the FFBD that do not represent system functionality and the arrows connecting them do not indicate direct operational flow. Instead, these data bubbles indicate what information is being transferred in between the functional blocks. They can also contain information about other exchanges such as resources or products that are transferred.

A similar function can be achieved by an arrow labels, but sometimes the information flow is between several functional blocks that may not be directly following the data’s origin functional block. Additionally, data bubbles are also commonly used to show elements that the system has to handle from outside of your system, whether it be something from perhaps another device that your system has to work with, information a user enters, or something in the greater context that your system has to deal with – context interactions can be quite varied and can range from things like inclement weather, a storage container, or in this case human sweat.

Formally acknowledging all of things that your system has to work with, that it may not have direct or any control over, can often mean the difference between a successful project and one that has to go “back to the drawing board” because the system’s inventors forgot to account for some real world condition that was outside of their initial score. For more on system context, check out the guide \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Formatting: Data information blocks are represented as rounded boxes or ovals. Either can be used, but only one type should be used throughout all of your FFBDs for consistency. The arrows connecting data information blocks should be solid arrows with solid arrow heads, like trigger arrows, however data information block arrows are always drawn on a diagonal to help distinguish them from trigger arrows.

**FAQ & Tips** **#1: Is there a right way to represent my system in an FFBD?**

There is more than one way to represent the same information using an FFBD! So don’t get hung up on having to do things the right way. Instead, focus on doing things the best way for you… What *IS* the best way for you? It’s however you and your team feel comfortable about understanding to enough detail how all of the parts of your system will operate together. As suggested in the Reference Blocks:Purpose section, sometimes it’s also helpful to arrange your FFBD in groups of function blocks that are the responsibility of certain teammate(s).

The first few times you do an FFBD its common to later realize that you and/or some of your teammates didn’t ask enough questions or request that more detail be added to the FFBD . So first off, don’t worry about appearing unintelligent, and it’s far better to ask questions now than work off incorrect assumptions and incomplete information. And secondly, don’t be afraid to add more things to the FFBD as you recognize things that you or others may have missed – and then be sure to tell everyone about it.

**FAQ & Tips** **#2: Can I repeat functional blocks?**

In some cases, you may need the same or very similar function done repeatedly in multiple places in your FFBD. There are several ways to handle this and they are demonstrated in the example.

* If the function is exactly the same and the output of that function is needed in more than one place, an edge shortcut is most commonly used.
  + An example is in the F.2.5 FFBD. Here, the “Estimate Sweating” output goes directly into “Set Microcontroller Settings” and edge shortcut “A”. Edge shortcut “A” then goes into “Measured Heart Rate” and “Measure Dissolved Oxygen” as if they were connected directly to the output of “Estimate Sweating”
* If the function is exactly the same but the inputs and outputs are different, the node can be repeated. The function block ID is also identical in both cases.
  + An example is in F.2 FFBD where “Create Health Report” is repeated. Regardless of what data is input into the function, the function of creating a report will be the same, even though the resulting outputs may also be different.
* If the function is similar but the inputs and outputs are different, AND what happens in the block is fairly different, the best practice is to give the blocks unique but similar names. The function block ID is also unique for the two block.
  + An example is in the F.2.5 FFBD. Here, the “Confirm Node Contact” function is repeated but differentiated by simply calling one version “A” and the other “B”. More descript names may be better but this may be okay assuming that the two sensors may have a remarkably similar “node contact” requirement, i.e. perhaps the mechanical solution for one may likely be similar to other’s – kind of like 2 different light bulbs that need a similar screw in receptacle.

You’ll notice words like “similar” or “fairly different” are used in the description above. As this implies, there has to be a judgement call made on your part – the key thing is that your naming conventions will assist you in your communications with others (see FAQ & Tips #3).

**FAQ & Tips** **#3: Does my FFBD have enough clarity?**

As a good check, ask yourself whether someone who picks up your FFBD for the first time a few months or even several years from now will be sure to interpret the FFBD correctly, without you being around to answer questions. If you’re hesitant that they will, that’s probably a good warning to you that you want to do something more to help clarify your FFBD for that new reader – and remember even you may feel like a new reader if you haven’t touched a part of your system for a while and you need to refer to your FFBD to remind yourself how things are supposed to work.

**FAQ & Tips** **#4: Formatting Tips for Building Your FFBD**

Although it’s common to want to build your FFBD to look neat and organized as you go, due to the natural iterative nature of FFBDs, this approach can actually be considerably more time consuming. Instead, it’s often better to roughly lay out the pieces before finally connecting any of them. Below are

1. Place boxes roughly where you think they would fit into the FFBD. Feel free to jot down potential arrow labels to get an idea of how much space they might take up but don’t draw any arrows yet. Also don’t worry about the function block IDs yet either.
   1. Much of the iteration should take place in this step. You may decide to split the FFBD into multiple referenced FFBD diagrams. Hence, you may even decide not to bother adding in the gates yet so long as you can keep track yourself of how everything should connect.
2. Place boxes into an organized layout, leaving enough room for arrow labels. Add in the gates, if you haven’t done so already.
3. Connect boxes & gates with arrows but do not adjust the arrows or arrow labels to look nice yet. It can also be easier to move around copies of formatted arrows (width, arrow head type, etc) rather than having to format arrows every time you draw a new one.
4. Once all arrows are placed adjust them and arrow labels to look organized
   1. As a suggested method of doing this:
      1. Start with one box at an end point of the FFBD.
      2. Adjust all arrows coming in and out of that box. You’ll notice that moving a box/gate on either end of an arrow in many software packages may reset the arrow to a default path between boxes – which is rarely the ideal path.
      3. Zoom in to see that the arrows are indeed straight.
      4. Select one of the boxes or gates connected to that first box and then adjust those arrows
      5. Repeat iii as needed.
      6. Tiny “kinks” along some mostly straight arrows is a common issue in PowerPoint. If you plan on sharing this with outside groups, it can be worthwhile to replace the “kinked” connector arrows with standard straight line arrows as is often used in the example.
5. Only once you’re satisfied with the diagram, set the IDs in the functional blocks.

It’s natural to feel some uncertainty – you’re learning a new language and hence being unfamiliar with the vocabulary can add a challenge to being able to express yourself as you desire.

**FAQ & Tips** **#5: Starting to Build Your System Implementation and Handling Uncertainty in Your FFBD**

Analyze the FFBD, look for critical functions Is there a path of limited operation that you can create, even skipping some functional blocks along the way, that will help you get to a prototype that you can test early on. Similarly, in creating your early prototypes, you may develop them in a more intelligent way to be expanded upon later because you recognize how they will need to integrate with more aspects of the system later on. Every system and project is different and there is no single right way as it will also depend upon the skill set of the team. Rank the blocks in terms of certainty. Those blocks that you may be the most uncertain may be some of the best ones to explore first, especially if they are critical to your overall operation & system performance – It’s far better to figure out early on that something won’t work or requires special needs to work properly, than to have developed the rest of the system that you are seemingly comfortable with only to later realize that it all needs to be redone to accommodate what you were uncertain about.

**FAQ & Tips** **#6: Using the FFBD to transition from the functional to the structural.**

The FFBD can be used to represent a structural solution system architecture as well as a functional one. Although Functional Flow Block Diagrams should in general focus on the functional aspects of a system architecture (as the name implies) it is not uncommon for a copy of an FFBD to be made later in the design process, where the functional block names are replaced with the decided upon structural solution.

An example of this is provided from an excerpt of another potential FFBD on slide 6. Notice that the swap between the functional and the structural does not need to be 1:1. But do notice that should you have multiple structural blocks to meet a single functional block, the structural blocks are all given the same functional ID number to help make the connection between functional and structural variations clear. As this kind of structural transformation is considered an informal technique, you may also see structural blocks with more than one functional ID in the blocker header as well, typically separated by a comma.

Overall though, since the original FFBD is still intact, the 2 FFBD variations together can serve as a way to map the structural solution to the function it is intending to meet. Although the structural versions are almost never submitted as part of an official government proposal, these more structural versions can sometimes make their way into externally released post-project documentation.