Regvue
Modern Hardware/Software Interface (HSI) Documentation

Rob Donnelly
robert.donnelly@jpl.nasa.gov

Josh Geden
joshua.m.geden@jpl.nasa.gov

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

I. Introduction

The current state of the art for register documentation is lacking. Register documentation is often in the form of a Word document, PDF, or static HTML. These mediums are highly static and can be difficult to share.

Want to automatically decode a register value into fields? These mediums won't help you. Want to share a specific part of the documentation with a colleague? With Word and PDF, you'll need to not only share the document but also the section number (and heading in case the number changes in the future). Your colleague will then need to download the document, make sure they have the necessary viewer installed, open the document, then manually navigate to the relevant section.

Regvue addresses all of these shortcomings and more. Regvue uses modern web-based technologies to provide a more rich register documentation experience.

II. Features

Regvue provides several features that optimize the register documentation experience. These features save time, reduce bugs, and reduce mental load. Figure 1 provides an overview of the major graphical elements of Regvue.

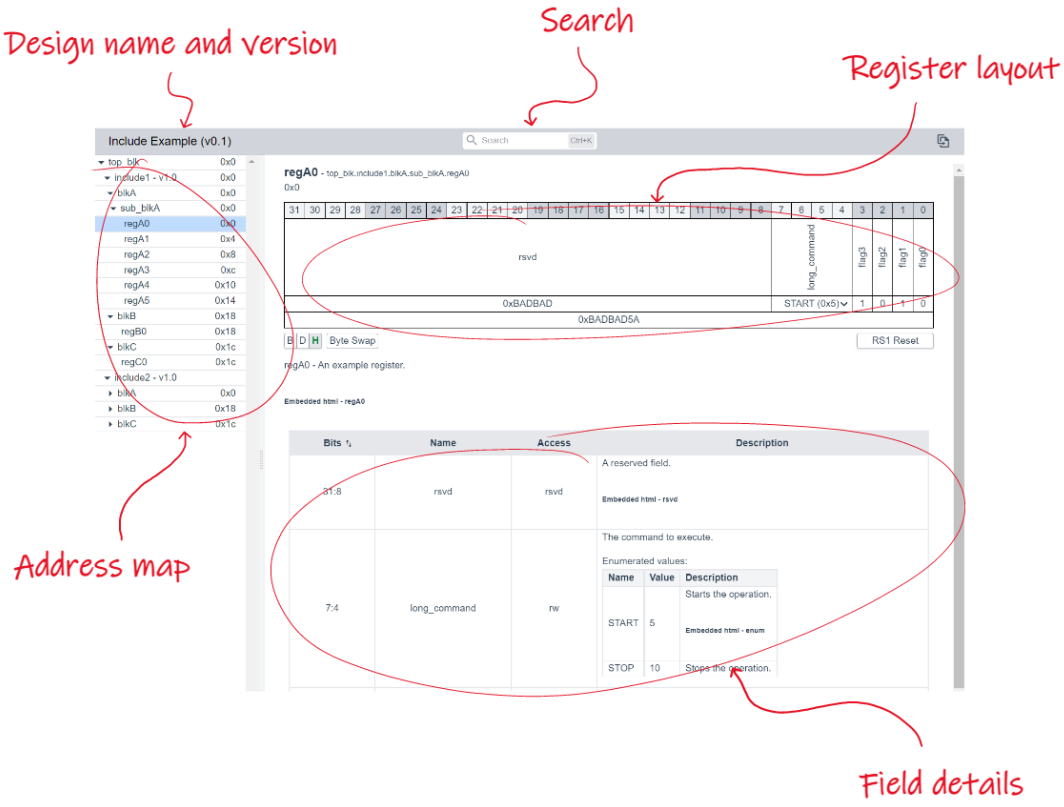

Figure 1. Regvue

Search

Regvue implements an incremental fuzzy search feature shown in Figure 2 that allows users to find the information they are looking for quickly all without leaving the keyboard. The search feature searches various attributes like names, descriptions, and offsets of various element types like blocks, registers, and fields. Search suggestions are updated and provided as the user types. Recent search selections are also presented to provide easy access to commonly referenced elements.

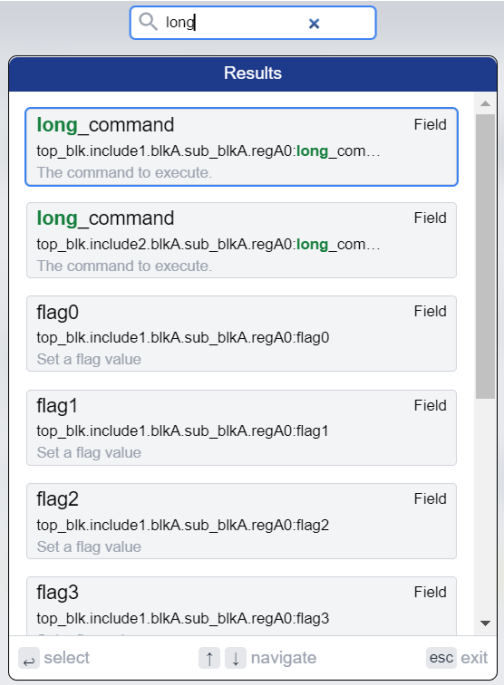

Figure 2. Search

Field/Register Encode/Decode

Regvue automatically encodes field values into register values and decodes register values into field values. For example, if the user wants to construct a register write value, they can enter the value they want to write to each field. The register value is automatically updated in real-time as the user enters each field value. The user can then copy and paste the register value to write. Similarly, if the user wants to decode a register read value into field values, they simply copy and paste the register read value into Regvue and it will automatically update the field values. Figure 3 illustrates how this works.

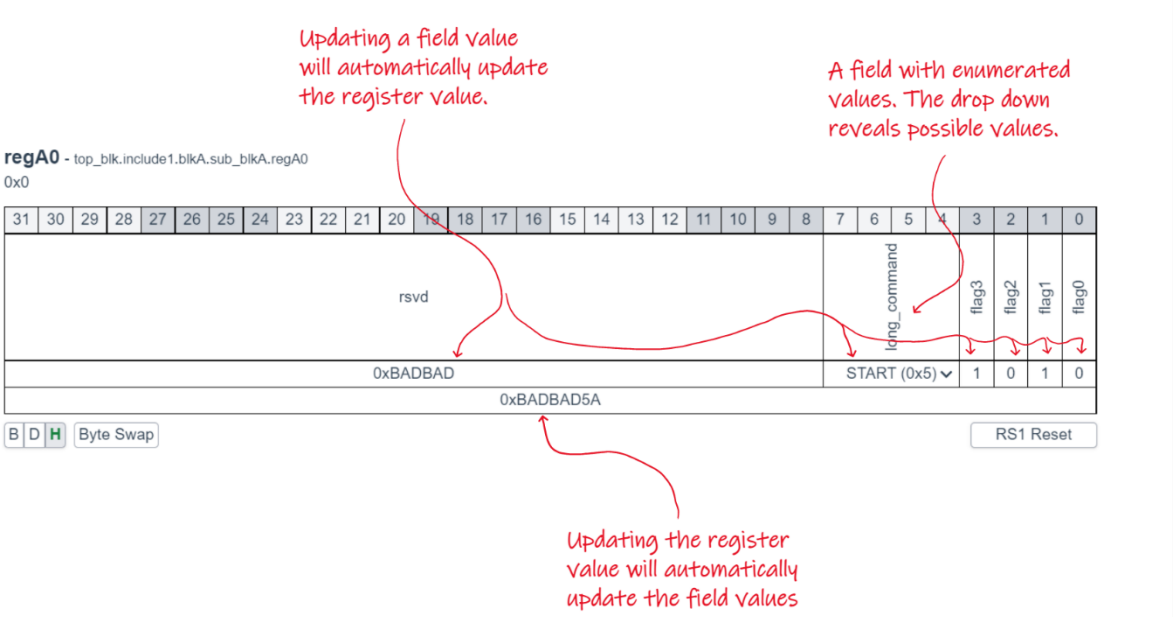

Figure 3. Field/Register Encode/Decode and Enumerated Values

Byte Swap

Regvue provides the ability to toggle byte swapping between field values and register values (and vice versa). This is useful when one venue (e.g. design and verification) uses little endian while another venue (e.g. software) uses big endian.

Enumerated Values

Regvue supports enumerated values which maps raw field values to more meaningful symbolic values and vice versa. For example, instead of seeing a value of "0x0" for the "state" field, the user will also see that it maps to a symbolic value like "IDLE". See the *long_command* field in Figure 3 for an example.

Multiple Bases

Regvue supports representing field and register values in binary, decimal, and hexadecimal. The base is selectable via the ‘B|D|H’ button group.

Multiple and Optional Resets

Regvue supports multiple resets per field. For example, a field may be affected by both power-on-reset and PCI reset while another field may not be affected by any reset. If a field is not affected by the selected reset, it will be shown as one or more question marks. An example of this is shown in Figure 4.

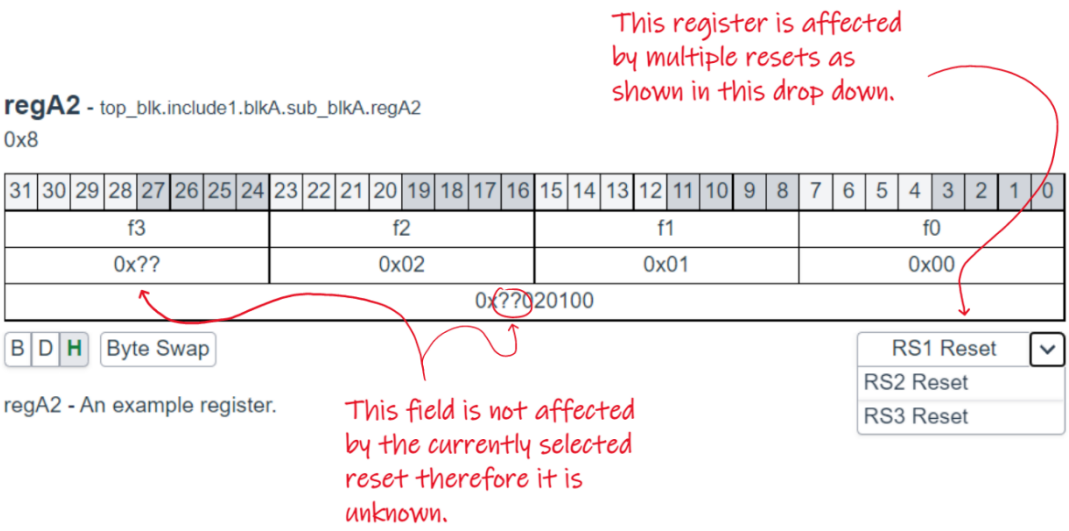

Figure 4. Multiple Resets

Direct Permalinks

Regvue provides permalinks or links that never change that link directly to various elements like blocks, registers, and fields. Gone are the days having to provide all of: a link to a document, a section number, and a section title.

Hosting

Regvue has minimal web hosting requirements. A basic static web hosting service like GitHub Pages or GitLab Pages is sufficient. It can even be run on localhost with a basic static web server like that provided by Python.

III. Usage

Input Format

Regvue is a design-agnostic web application. Design-specific register information is loaded from one or more JSON (JavaScript Object Notation) files (see Listing 1 for a small example) that use the Register Description Format (RDF) schema. The RDF schema was created specifically for Regvue. RDF takes some inspiration from the CMSIS-SVD (Common Microcontroller Software Interface Standard - System View Description) file format [1]. The motivating factor behind RDF was to minimize the amount of post processing necessary. Regvue uses JSON because Regvue runs on a JavaScript engine and JavaScript has native support for deserializing JSON into an object. Using an XML based format like CMSIS-SVD or IP-XACT [2] would have required more complex XML parsing using the DOMParser [3] Web API. Using a JSON based format makes parsing a single call to JSON.parse() [4] JavaScript API.

RDF JSON supports including other RDF JSON files. This enables higher-level views to be constructed out of lower-level views. For example, a system-level RDF JSON can include multiple chip-level RDF JSONs which can in turn include multiple module-level RDF JSONs.

```
{
  "schema": {
    "name": "register-description-format",
    "version": "v1"
  },
  "root": {
    "desc": "Example Design",
    "version": "v1.0",
    "links": [
      { "text": "GitHub", "href": "https://github.com/org/repo" }
    ],
    "children": [
      "reg0"
    ]
  },
  "elements": {
    "reg0": {
      "type": "reg",
      "id": "reg0",
      "name": "reg0",
      "offset": "0",
      "fields": [
        {
          "name": "f0",
          "lsb": 0,
          "nbits": 32,
          "access": "ro",
          "reset": "0"
        }
      ]
    }
  }
}
```

Listing 1. A small but valid RDF JSON document

Schema

The RDF JSON format is described using the JSON Schema [5] standard. Regvue uses the RDF JSON schema to validate RDF JSON documents. For example, if a required property is missing, the validator will report an error and Regvue will present the error to the user. RDF JSON writers can also use the RDF JSON schema directly (without Regvue) to validate their RDF JSON generators.

The RDF schema includes schema versioning information to enable making breaking changes to the schema in a backwards compatible manner. For example, a future version of the schema could add a required property. Existing RDF JSON documents would be incompatible with this new schema but since every RDF JSON document contains schema version information, Regvue could support both the old and new version of the schema and thus support RDF JSON documents that target either version.

Web Application

The Regvue web application instance is deployed to a static web hosting service by simply copying the necessary files to a directory. The instance is now available via a URL (e.g. https://example.com/regvue). A single deployment can serve arbitrarily many designs. A user can load the Regvue application by navigation to the instance URL. The user will be prompted to load an RDF JSON as shown in Figure 5.

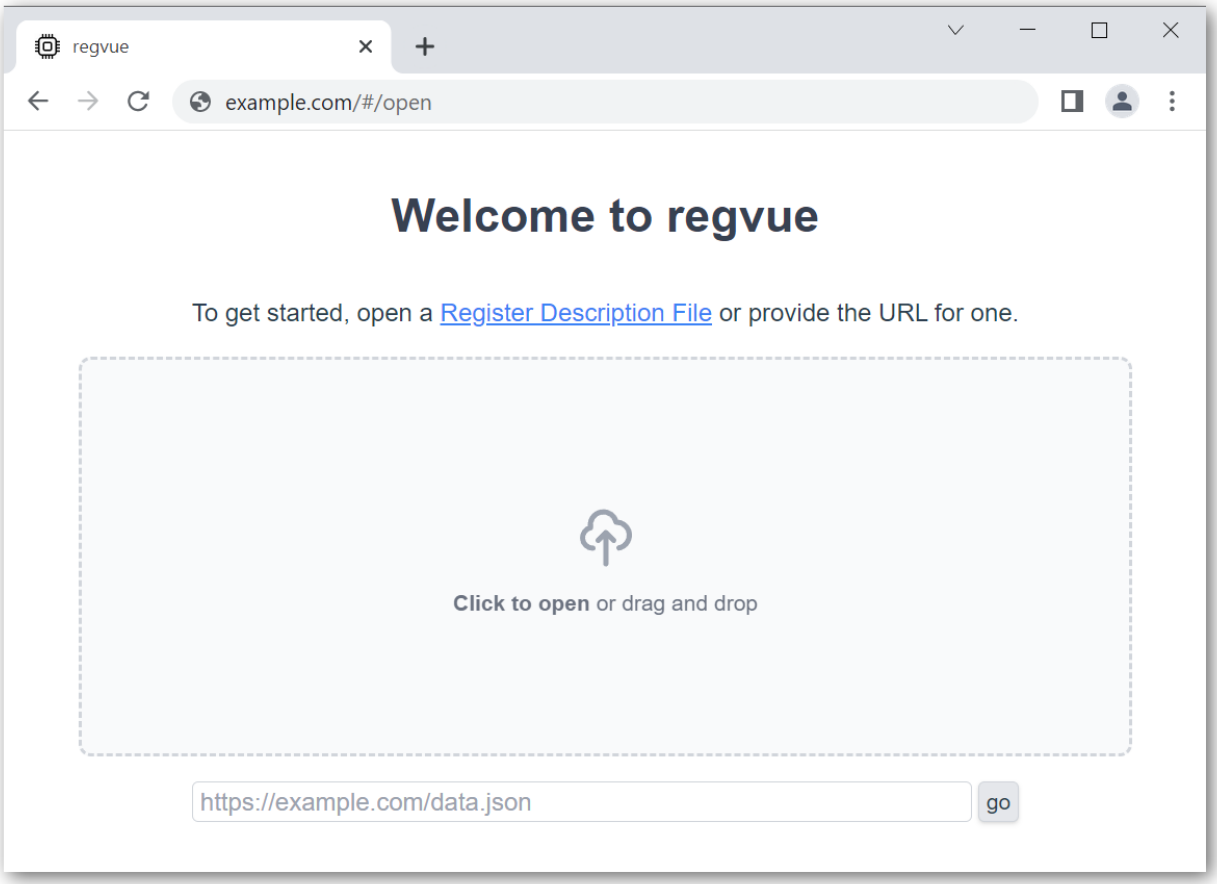

Figure 5. Prompt to load an RDF JSON

The user can then load an RDF JSON from a URL or via the filesystem. The URL method is preferred because it supports includes and is more easily shared with others. Loading via the filesystem does not support includes because of the how file origins [6] are treated by the Same Origin Policy, a security policy implemented by modern web browsers. After loading an RDF JSON via a URL, Regvue will display the design home view and update the browser URL to a permalink that includes the path to the RDF JSON. The user can then copy and share this URL with others to take them to the same view. Figure 6 shows a permalink that directly opens the *regA1* register from the https://example.com/data.json RDF JSON file in Regvue at https://example.com.

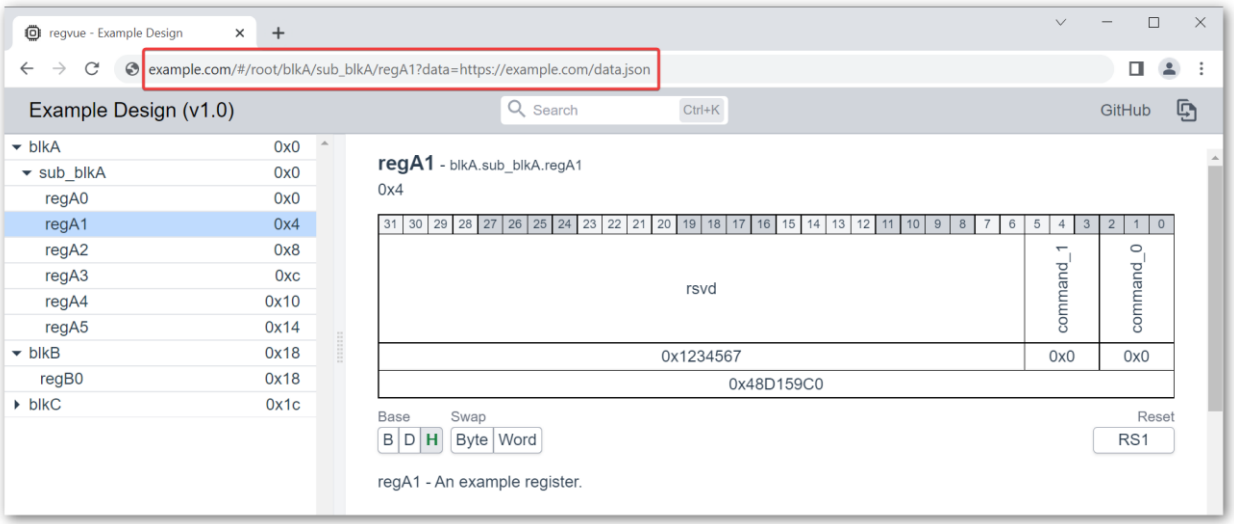

Figure 6. Permalink to RDF JSON

Hosting Considerations

At JPL, we use GitHub Enterprise to host both the Regvue application and the Regvue JSONs. More specifically, the Regvue application is hosted on GitHub Pages and the Regvue JSONs are hosted directly in Git repositories on GitHub Enterprise. The Regvue team maintains the Regvue application instance on GitHub Pages. Regvue users are encouraged to use this instance but are free to install and maintain their own. Users can generate their Regvue JSON(s) then commit and push them directly to Git. This has three benefits and one drawback. The first benefit is that this makes it really easy for users to host Regvue JSON, they simply push to GitHub and load the GitHub raw link for the Regvue JSON into the Regvue Application. Users do not need to procure web hosting nor do they need to know how to use GitHub Pages. The second benefit is that users get versioning for free. The Git version information is embedded in the URL of the Regvue JSON. If users want to load a different version, they simply modify the URL. The third and final benefit is that users also get authentication for free. Access to the Regvue JSON is controlled by normal GitHub access controls. The drawback of hosting the Regvue JSON directly in Git is that security policies of modern browsers and GitHub means that the Regvue application must be hosted on GitHub as well. Regvue uses the browser Fetch Web API. This API obeys the Single Origin Policy which makes it so that you can only fetch things from the same domain or origin. For example, if Regvue is on GitHub, it can only fetch URLs that are also on GitHub. Or to put another way, to fetch a URL from GitHub, the fetch must occur from GitHub. The Cross-Origin Resource Sharing (CORS) standard provides a way to relax the Single Origin Policy. However, the CORS configuration for GitHub raw links does not.

IV. Usage on Europa Clipper

Regvue was developed and used on the Europa Clipper [7] mission. Europa Clipper is a spacecraft that will study Europa, an icy ocean moon of Jupiter. An existing in-house register automation tool called Register Translator was updated to generate the RDF JSON used by Regvue. Register Translator was then used to auto-generate the RDF JSON for most of the FPGA designs on Europa Clipper. Two FPGA designs required writing ad hoc Python scripts to parse FPGA design specifications that were not compatible with Register Translator. A top-level RDF JSON for the spaceflight computer or Europa Compute Element (ECE) was created with a simple shell script. The top-level ECE RDF JSON simply includes the FPGA-level RDF JSONs to provide a system-level view of the spaceflight computer. Figure 7 shows the process for generating the RDF JSON for Europa Clipper.

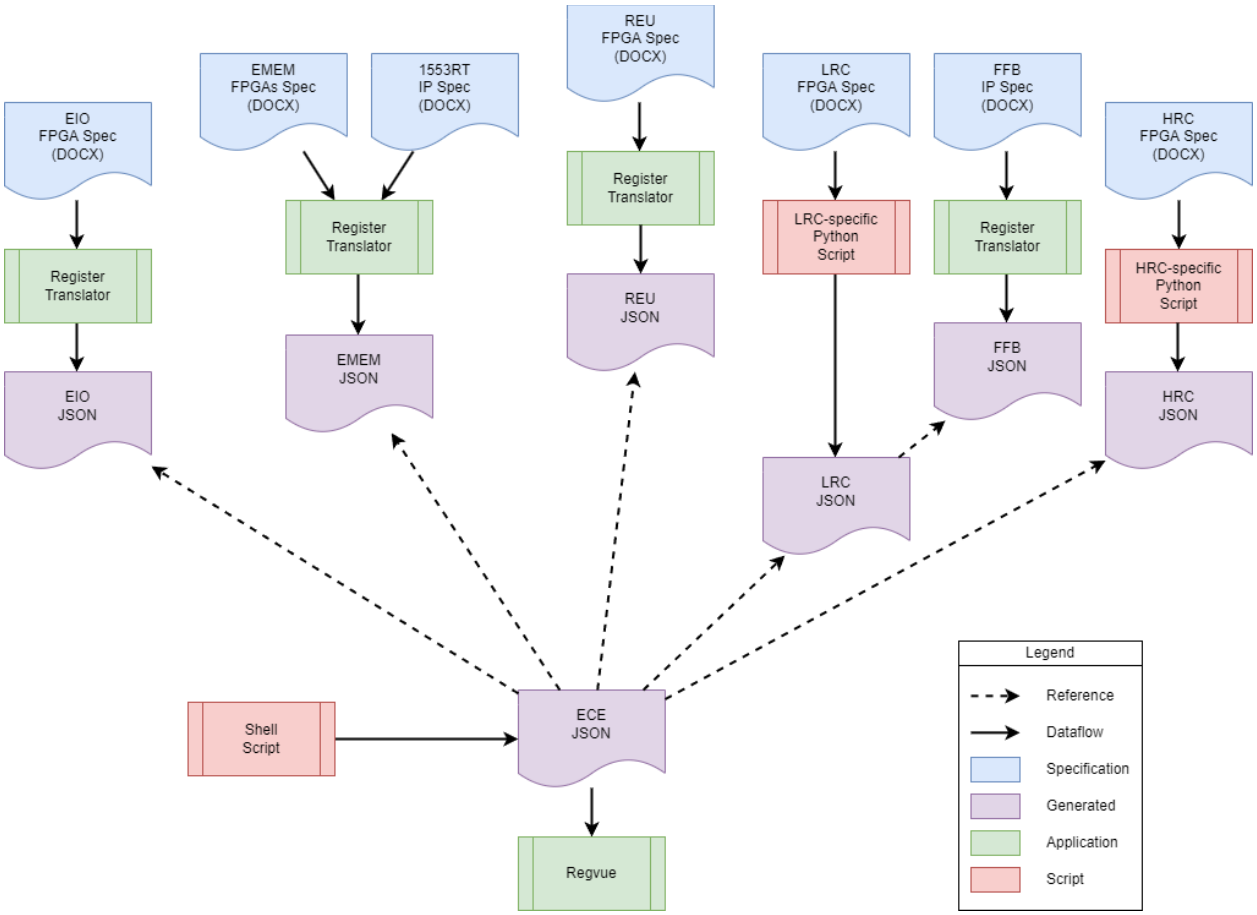

Figure 7. Generating the RDF JSON for Europa Clipper

V. Implementation

Open Source

Regvue is currently in the process of being released as open source under the Apache 2.0 and/or MIT licenses. When Regvue has been released as open source, it will live under the NASA-JPL GitHub organization [8].

Technologies

Regvue is implemented in TypeScript [9] and the Vue [10] frontend framework.

TypeScript is basically JavaScript with types. It compiles down to JavaScript. TypeScript was chosen to both catch more bugs at compile time and to improve maintainability. With dynamically typed languages like JavaScript, you might write a function that takes an array as an input. If you later call this function and pass a string, JavaScript will throw a run time error. Ideally this is found during test otherwise the error may be seen by the user. In TypeScript, such a mistake would be found at compile time.

A frontend framework like Vue allows frontend web applications to be written at a higher level of abstraction than raw HTML + CSS + JavaScript. Vue has the concept of a component which encapsulates a UI element. In Regvue for example, the search functionality is a component, the address map is a component, etc. Components can be instantiated inside of other components. In Regvue for example, the register layout component is composed of several other components for the field labels, field values, and register value. The component abstraction makes it a lost easier to implement complex behaviors.

Prior Work

The idea for Regvue originated from a desire to graphically peek and poke registers in live hardware. You'd be able to explore the design much like the File Explorer application in Windows. We called it Register Explorer. We started prototyping the GUI for Register Explorer. The first prototype was a desktop application written in Python and the Tk GUI toolkit. The register layout view is the most complex graphical portion of the application. It was the piece that would make or break the GUI so it was always the piece to be implemented first. Implementing the register layout in Tk required too many hacks and didn't work very well. It was at this point a realization was made. Register information is documentation. Trying to hand implement a render for documentation was silly. The best solution for

interactive documentation today is HTML. A small prototype of the register layout was created in raw HTML + CSS + JavaScript that implemented dynamic rotation of field labels depending on field name length and table cell width. This worked very well and confirmed that we were on the right path. A third prototype shown in Figure 8 was created to integrate the HTML prototype into the Python Tk GUI. The idea was to replace the custom Tk-based register layout rendering with an embedded web browser. This too worked quite well.

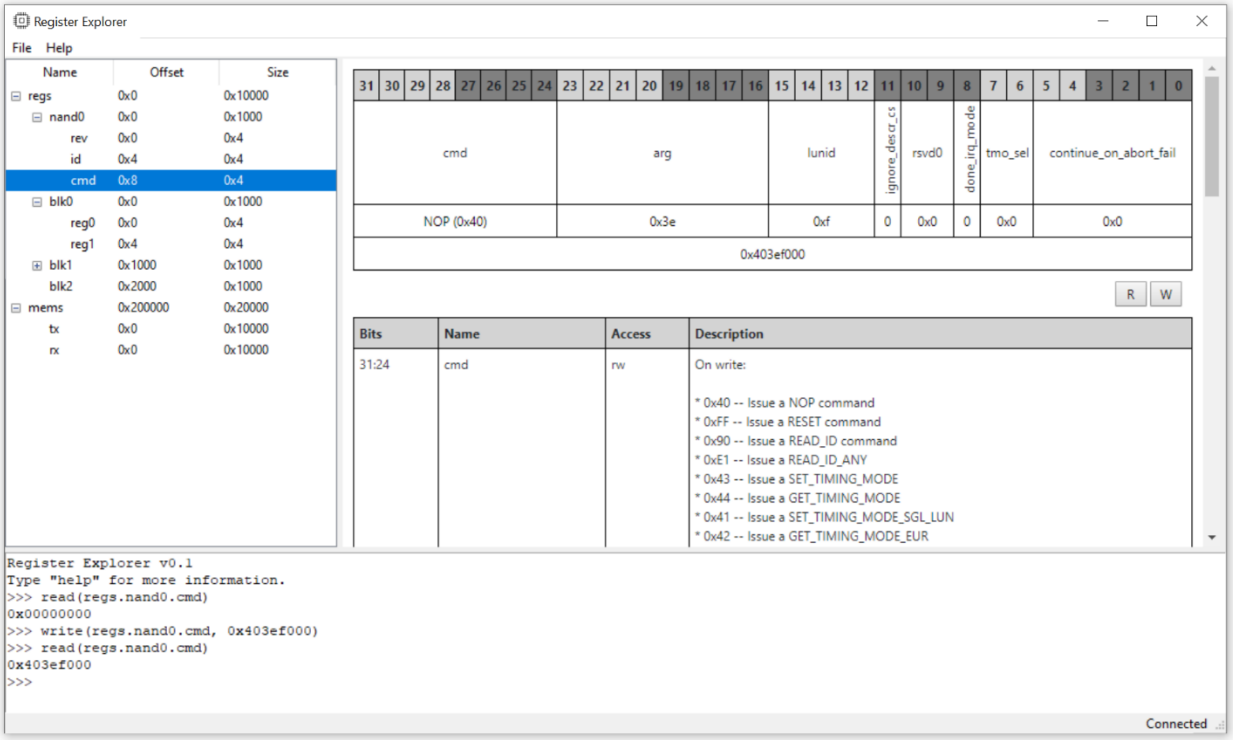

Figure 8. Register Explorer prototype with Tk GUI and embedded web browser

The problem however was packaging and support of the embedded web browser. We used the cefpython library which provides Python bindings for the Chromium Embedded Framework which effectively allows the Chrome web browser to be embedded in an application. We needed to support the 3 major operating systems and create installation packages for each. We were unable to find a turn-key solution to do this for cefpython. Additionally, cefpython is unevenly supported due to the lack of consistent funding.

The path forward was now clear. We'd make it a web app. It could then be accessible by any web browser and we could use something like Electron [11] to deliver a desktop version with additional functionality like interacting with live hardware. Rob created a very rough prototype of Regvue, demoed it internally at JPL, and secured funding to hire Josh as a summer intern to make Regvue what it is today.

VI. Future Work

The original desire was for a GUI to manipulate and explore registers in live hardware. We now have a web application but no desktop application. As mentioned previously, something like Electron could be used. The problem with Electron however, is that the package size is so large. Every install ships with a full version of the Chromium web browser which makes every Electron application 100+MB. Enter Tauri [12]. Tauri, like Electron, allows desktop applications to be built using web technologies. Unlike Electron however, Tauri uses the web view provided by the operating system. Because of this, Tauri applications are much smaller, 1 to 2 orders of magnitude smaller. We've created a working prototype of Regvue Desktop shown in Figure 9 that can read and write registers over a UART serial console. However, there is still much to figure out before it can be deployed let alone made public.

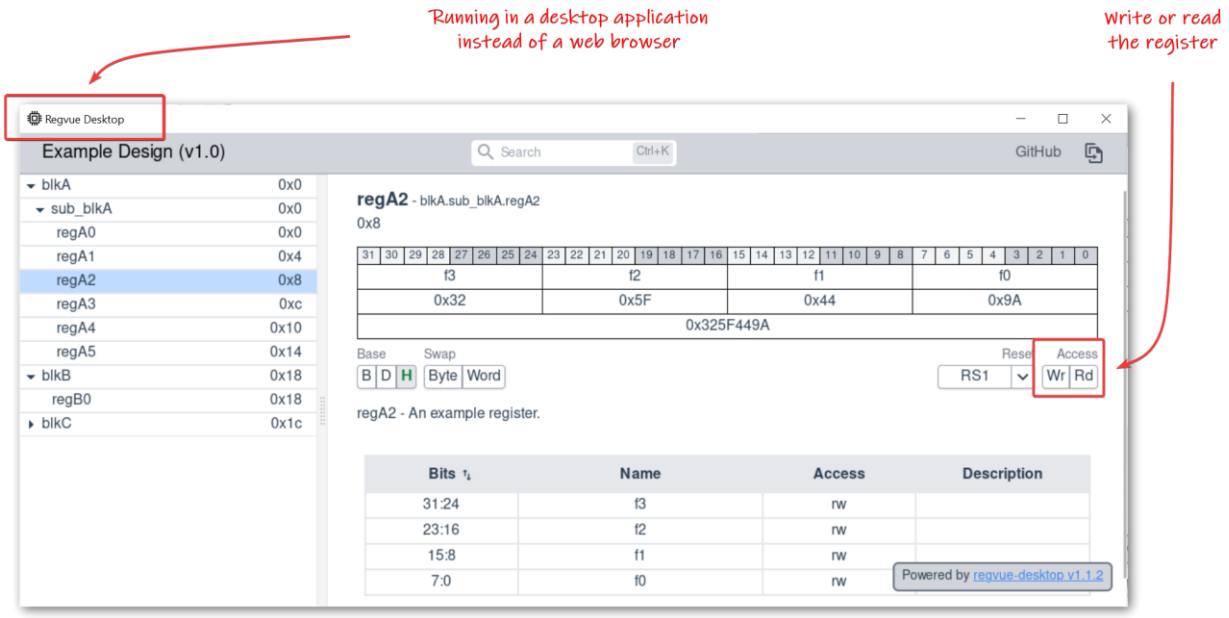

Figure 9. A working prototype of Regvue Desktop

ACKNOWLEDGEMENTS

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

REFERENCES

[1] Arm Ltd, "CMSIS System View Description," https://www.keil.com/pack/doc/CMSIS/SVD/html/index.html
[2] Accellera, "IP-XACT," https://www.accellera.org/downloads/standards/ip-xact
[3] Mozilla, "DOMParser," https://developer.mozilla.org/en-US/docs/Web/API/DOMParser
[4] Mozilla, "JSON.parse()," https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse
[5] A. Wright, H. Andrews, B. Hutton, G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents," https://json-schema.org/draft/2020-12/json-schema-core.html
[6] Mozilla, "Same Origin Policy – File Origins," https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#file_origins
[7] NASA Jet Propulsion Laboratory, "Europa Clipper", https://www.jpl.nasa.gov/missions/europa-clipper
[8] NASA Jet Propulsion Laboratory, "NASA-JPL GitHub Organization," https://github.com/nasa-jpl
[9] Microsoft, "TypeScript", https://www.typescriptlang.org
[10] Evan You, "Vue", https://vuejs.org
[11] OpenJS Foundation, "Electron", https://www.electronjs.org
[12] Tauri Contributors, "Tauri", https://tauri.app