Collaborative Circuit Designs using the CRAFT Repository

Adam Brinckman[a], Ewa Deelman[b], Sandeep Gupta[c], Jarek Nabrzyski[a], Soowang Park[c], Rafael Ferreira da Silva[b], Ian J. Taylor[a,d], Karan Vahi[b]

[a] *University of Notre Dame, Center for Research Computing, Notre Dame, IN 46556, US*
[b] *Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA*
[c] *Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA*
[d] *Cardiff University, School of Computer Science & Informatics, 5 The Parade, Cardiff CF24 3AA, UK*

Abstract

This paper provides an overview of the CRAFT repository, which exposes a collaborative gateway enabling circuit designers to share methods, documentation and intellectual property. The main goal for the repository's development is to ensure that future designs for custom integrated circuits need not be reinvented for each design and fabrication cycle. This paper presents the architecture, design, and implementation of the collaborative repository, which capitalizes on the recent advances in production quality open-source collaborative framework, which are interfaced using a lightweight Javascript front-end to satisfy the requirements of the DARPA's CRAFT program. The repository has been developed as an EmberJS application (front-end), that interacts with an instance of the Open Science Framework (OSF). This paper contextualizes the framework from the viewpoint of circuit designers and outlines the advantages of the tools and visualizations offered by the repository, in terms of increasing the efficiency of designers' tasks. To this end, we also provide a description of two specific tools that have been exposed using the repository, which build on JSON schemas and allow users to develop and visualize the circuit design flow diagrams and the intellectual property they can reuse to accelerate their design process.

Keywords: Collaborative Environments; Design Flows; Chip Design

1. Introduction

ASIC designers typically have years of formal training and experience, and hence a significant understanding of the prevailing design tools, the formats/languages used to describe various artifacts created during the design process, typical sequences in which the tools are used (called design flows) to achieve typical design requirements, and of available intellectual properties (IP, also called IP cores), which they use to reduce design time and

Email addresses: abrinckm@nd.edu (Adam Brinckman), deelman@usc.edu (Ewa Deelman), sandeep@usc.edu (Sandeep Gupta), naber@nd.edu (Jarek Nabrzyski), soowangp@usc.edu (Soowang Park), rafsilva@isi.edu (Rafael Ferreira da Silva), ian.j.taylor@nd.edu (Ian J. Taylor), vahi@isi.edu (Karan Vahi)

costs. ASIC designers' responsibility encompasses understanding the specifications for an Application Specific Integrated Circuit (ASIC) under design, including the functionality the ASIC must implement and constraints on values of its key parameters, such as speed and power consumption; and the use of available tools, IP, and design flows to carry out design, analysis, and simulations to obtain ASIC designs that meet the desired specification, while minimizing design costs and time to market.

The Circuit Realization at Faster Timescales (CRAFT) program by the Defense Advanced Research Projects Agency (DARPA) [1] aims at reducing the design cycle time to market (referred to as *timescale* in the program) for creating custom ASIC integrated circuits. Today, ASIC designs can take more than two years, require a large team of engineers, and incur costs well in excess of $100 million. Such timescales and economics are not practical and therefore Department of Defense (DoD) engineers tend to use readily available inexpensive general-purpose chips (field-programmable gate arrays (FPGAs) and general purpose processors(GPPs)) and implement the specialized operations using reconfiguration (of FPGAs) or software (on GPPs). The resulting chips require more power, which is not ideal for hand-held devices that are deployed in the battlefield or for use in unmanned aerial vehicles.

CRAFT's program goal is to reduce the timescale for designing power-efficient high performance ASICS for military applications to months. (In this paper, we use *chips* and *ASICS* interchangeably.) In order to facilitate this goal, the CRAFT repository also aims to provide innovative tools so that methods for fast design, documentation, and intellectual property can be re-purposed, rather than re-invented, with each design and fabrication cycle.

Our team has been working on designing and building the CRAFT repository [2], with the aim of it being a collaborative gateway for circuit designers to be able to work on chip designs together and share their methods, tools, and designs. For developing this repository, we have taken a hybrid approach by integrating a CRAFT-specific Web front-end, written in EmberJS [3], with the preexisting Open Science Framework (OSF) [4]. We make use of the OSF's REST API for the integration. We have further reused some of the Web graphical interfaces by virtual co-locating of an OSF server instance alongside our customized CRAFT Web application. This approach involved some effort in the minor customization of the OSF instance e.g., LOGO, titles, etc., but also allowed us to include several pieces of functionality without modification (e.g., file browser and visualizing, adding contributors, a WIKI, and so on). The CRAFT application implementation was confined to CRAFT-specific additions to support the collaborative tools for chip design and the remaining parts of the OSF seamlessly work together in an extremely interactive on-line experience.

This paper describes the architecture, design, and implementation of the CRAFT repository and it's collaborative tools. This work is far more comprehensive than our previous publication [5]. Here, we focus far more towards the user and to enable circuit design reuse. We also extend this previous work by describing the new CRAFT specific extensions to this repository that allow a user to specify, edit and visualize their design flows and their description of their intellectual property surrounding their design. Therefore, we added two new sections in Section 6 that focuses on the Craft tools to enable collaborative circuit design. Such tools enhance the overall chip design process by facilitating re-use of circuit modules

and rapid adoption of new design flows across CRAFT program participants.

The rest of the paper is organized as follows. Section 2 provides a typical workflow for a circuit designer and how their scientific methods can be improved through the use of more advanced tools. Section 3 discusses the requirements for the CRAFT repository and extracts the common primitives and constructs needed for implementation. Section 4 discusses the architecture and design of the craft repository and Section 5 discusses the core implementation of the repository. Section 6 describes the collaborative tools, which allow a Craft user to enter their design flows and IP. Section 7 discusses future directions, and Section 8 concludes the paper.

2. CRAFT Impact for the Collaboration of Circuit Designers

ASIC designers must have a deep understanding of available tools, formats/languages used to represent the artifacts and information created during design, design flows, and IP. Traditionally, all this information is captured in comprehensive manuals and supported by training material. Traditional formats and comprehensiveness of these manuals necessitates years of formal training and experience for new entrants. However, in normal circumstances this works reasonably well, since a designer must face the difficult learning curve only once, when they first enter the ASIC design field. This is due to the fact that they can subsequently update their knowledge base incrementally, since the tools, languages, flows, and IP change slowly.

In contrast, the DARPA CRAFT program poses major new challenges due to two reasons. First, CRAFT plans to develop a wide range of radically new tools, languages, flows, and IP, necessitating significant learning even for experienced ASIC designers. Second, one key goal of this program is to provide a path for DoD designers with little prior ASIC design experience to design ASICs using the new methods and tools developed by the CRAFT performers. Simply, this necessitates us to develop new ways to represent the knowledge, especially about design flows and IP, in such a way that an engineer new to ASIC design can learn and start using these quickly. Special challenge is posed by the design flows, since these are typically described using scripts which are difficult to understand, must be adapted to each design team's specific objectives and computing environment, and must be used in iterative ways that currently rely completely on designer's experience.

We are creating our repository to provide the support the achievement of the above objectives of CRAFT. Specifically, our repository provides support for communities of designers, includes new ways to represent knowledge about tools and languages as well as completely new ways to represent and adapt design flows, and a new approach for submission, search, and use of IP.

Leading vendors of ASIC design tools provide frameworks that consolidate the key steps of ASIC design and verification, by providing lists of these steps, managing the status of each step, and displaying key results from each step. The Lynx [6] system from Synopsys is an example of such a framework.

In contrast to these, our project addresses the unique challenge created by the ambition of the DARPA CRAFT program. On one hand, CRAFT program is supporting the creation

of several completely new tools and languages to be added to existing commercial repertoire. On the other hand, starting in its Phase 2, CRAFT program is planning to bring DoD design teams, some with little prior ASIC design experience. Our repository's goal is to provide an environment where design teams with little prior experience are able to learn a vast array of new tools and languages and to provide support that will enable them to harness new and old tools to design ASICs in highly reduced timescales. In the first phase of the CRAFT program we have focused on the development of tools to capture the development and description of design flows and IP cores (presented in this paper). In the second phase (recently started), we aim to develop tools to enable the discovery of these contributions as well as to facilitate the understand and execution of the process to use the IP cores or execute the flows.

3. Craft Requirements

To derive the key characteristics of the repository, we began by analyzing the goals of the CRAFT program. First and foremost, CRAFT requires the development of radically new chip design approaches that will dramatically reduce the time required for the first successful tape-out[1] for ASIC designs. The community of researchers and industry experts working to address this challenge will develop completely new tools, new types of IP libraries, and new design flows. Second, the program requires CRAFT researchers to transfer immediately all their new tools, IP libraries, and desing flows to a few selected DoD ASIC design teams, and requires these teams to use such tools to design ASICs. Third, the program requires rapid refinement and adoption of the new tools, libraries of module designs, and design flows by the wider community of DoD ASIC design teams.

The CRAFT repository therefore must expose many of these tools to help the CRAFT community organize information and to allow the different teams to communicate efficiently with each other. At the same time, the various teams need to have control over their content, hence the repository's architecture must be sufficiently flexible to provide designers the ability create different permissions for different projects and topics. The following five sections discuss the underlying requirements and the high level concepts we use to attain CRAFT goals.

3.1. A Collaborative Space For Chip Designs

The CRAFT project centers collaboration around a design flow. Therefore, the repository requires a way to encapsulate the tools within some kind of collaborative space. To meet this requirement, we decided to organize the repository using the concept of a *project*, in a way very similar to other on-line tools, such as Bitbucket [7] and GitHub [8]. Each project should have a project overview page, which provides access to the ensemble of features of the project, as well as an overview of the components, files, tags, history, comments, and other components (e.g., wikis associated with the project). Participants should also be

[1] In the circuit design context, tape-out is the final result (or the resolution of the cycle) of the design process for integrated circuits (ASICs).

able to participate in discussions by leaving comments within a project, effectively creating a project-wide chat. Further, notifications should be enabled for any project, allowing users to be notified via email when a new comment is added to a project. A user can also choose to receive email notifications when someone replies to their comment, similar to how they would do on Facebook.

Each project should be able to provide documentation (e.g., a wiki), describe itself (tags), and allow files to be uploaded and browsed. Sub-projects should be allowed and should be distinct, encapsulated parts of a project and have their own contributor lists and permissions. For example, the user should be able to create a data component that remains private even when other parts of the project are made open to other collaborators, and list contributors that were vital for data collection but are not involved in other parts of the project. Thus, projects have a hierarchical collaborative space.

Each project therefore should provide a container to allow a user to organize files and content into meaningful groups like catalog of design flows, standard reference flows, proposed flows, datasets, code, circuit modules and IP modules designed, or other research contributions. Each project should also have a unique, persistent URL, meaning that it can be referenced or linked to individually. Every action/event should be automatically documented with date-time stamps, and the log presented on the project dashboard. Additionally, a project should be capable of being completely open or private.

3.2. Authentication

Users should authenticate via a secure central authentication service (CAS). The CAS protocol involves at least three parties: (1) a client web browser, (2) the web application requesting authentication, and (3) the CAS server. It may also involve a back-end service, such as a database server, that does not have its own HTTP interface but communicates with a web application. When the client visits an application desiring to authenticate to it, the application redirects it to CAS. CAS validates the client's authenticity, usually by checking a *username* and *password* against a database (such as Kerberos, LDAP, or Active Directory). If the authentication succeeds, CAS returns the client to the application, passing along a security ticket. The application then validates the ticket by contacting CAS over a secure connection and providing its own service identifier and the ticket. CAS then gives the application trusted information about whether a particular user has successfully authenticated. CAS allows multi-tier authentication via proxy address. A cooperating back-end service, like a database or mail server, can participate in CAS, validating the authenticity of users via information it receives from web applications.

3.3. Project Permissions, Communication, and Privacy

Each project should have a project leader (project administrator, e.g., the PI could be a project administrator), who can manage that particular project and its sub-projects. The project administrator should be able to invite collaborators to their project and assign them certain privileges. Additional roles include: (1) `read` privileges, allow contributors to see the contents of the project or component; and (2) `read and write`, which allow the contributor to see the contents of the project, upload and delete files, create, and edit new

projects. Administrators encompass `read and write` privileges, as well as the ability to add or delete contributors, controlling permissions, and controlling the overall settings of the project, or even deleting the project.

All projects should be private by default. However, a project administrator can choose to make a project publicly available. However, projects that are public can still contains sub-projects with their own privacy settings, i.e., making a project public does not make all of its sub-projects public. Users may want to provide more limited access to external users for a variety of circumstances. For example, editors or reviewers might get read-only access to a private project during the review process.

3.4. Version Control

The repository should support version control as part of its services. Project members can keep things up to date by uploading new versions of documents to the repository and have the repository keep track of older versions of those documents.

We also plan to support "releases", which create a frozen, time-stamped version of a project that cannot be edited or deleted. The release will have its own unique, persistent URL that is always linked back to the *frozen* project. The meta-data should be permanently stored with a registration.

3.5. Visualizing and Editing Design Flows

The design flows should be captured using a machine processable format, which is capable of being visualized by Web-based tools. From our initial requirements gathering, our users have indicated that a graph view and a table view are the two initial visualization tools we should provide for the design flows. For editing the flows, initially these will be input manually but in the second phase of the project, we target the support of editable components via the table or graph viewers.

4. Architecture and Design

Our architectural design approach in developing a repository consistent with the requirements of the CRAFT program, was guided by the principles of optimizing development and enabling sustainability. We identified four candidate approaches that would enable us to converge to an efficient system: 1) architecting and implementing the system from scratch; (2) customizing an existing system to meet the needs; (3) creating a new dashboard to an existing system using a REST API; or (4) creating a dashboard for CRAFT-specific features to an existing system using a REST API, and leveraging existing tools using a hybrid architecture.

The first option is essentially a straight forward engineering task, considering the fact that the requirements are well defined. It provides us certain advantages such as no other systems to learn and integrate with. However, the development manpower available to us (1.5 full-time developers) and the timescale of 7 months, from project requirements gathering to a first production release of the repository made it infeasible. Another drawback is that a subset of the features already exist on other websites; e.g., Github and Bitbucket. They already have

many of the project-oriented features implemented, amongst a number of others. That in itself, made this approach a poor case of reinventing the wheel. Lastly, sustainability is not guaranteed and would have required on-going additional effort to maintain the code base.

Given the above considerations we then decided to investigate existing systems that may satisfy some or all of the CRAFT requirements. We identified two candidate systems that were then investigated in depth. We discuss our findings and analysis below.

The Open Science Framework (OSF) [9, 4] is an open source software project that facilitates open collaboration in science research. It focuses primarily on the reproducibility of scientific research. OSF has a notion of a project (called a node or component in OSF), that allows users to invite other people to collaborate. Administrator of a project can assign various privileges to users such as administrative, read and write. The system natively supports file versioning, allowing users to upload new versions of a file and keep the same name. Users can see all the versions of the files. At the project level OSF supports forking and registration of a project. Registration of a project results in an immutable release that is good for sharing working functionality with the larger community. For importing and exporting data into the repository, OSF additionally interfaces with several popular cloud-based storage systems, e.g., Dropbox [10], Google Drive [11], and Box [12]. Projects in OSF have a full WIKI capability for integrating documentation, a facility for adding comments and tools for visualizing different types of files. OSF has recently evolved into a more general framework for enabling project-based functionality. It provides a JSON-API [13] compatible REST interface for all of the functionality. The API is provided using the Django Rest Framework [14]. OSF also provides an EmberJS [3] toolkit that allows third party implementations to reuse the models of the OSF API from a Javascript application.

From a sustainability point of view, both systems offer good support. HUBzero provides the HUBzero Foundation, which is a community-based, non-profit organization that promotes the use of HUBzero and ensures ongoing sustainability of the core software. The OSF has various methods to allow people to contribute to the longer term sustainability of the software, including donations using Crowdrise or other forms of payment. The OSF has also set aside a fund to continue the support for the framework.

HUBzero [15] is another existing system that addresses most of our requirements. It is an open source software platform for building websites that support scientific activities. The Hub excels in providing interactive graphical simulation tools. For example, you can zoom in on a graph, rotate a molecule, probe isosurfaces of a 3D volume interactively. The Hub also provides a collaboration space, where users can come together and share information. Fine grained controls exist for managing group information, such as allowing the creator of a group to accept or reject members. Projects can be made private limiting access only to registered members of the group. The Hub has a notion of "topic" pages, that are similar to the Google "knol" model for knowledge articles.

Both HUBZero and OSF support most of the features that CRAFT requires. One of our main motivations with the development of this repository, was to keep the CRAFT web dashboard separate and decoupled from the existing systems code bases. Although HubZero has good interfaces in PHP available, it lacks a comprehensive Web API that can be used to separate the HUBzero instance from the CRAFT web dashboard. On the other hand,

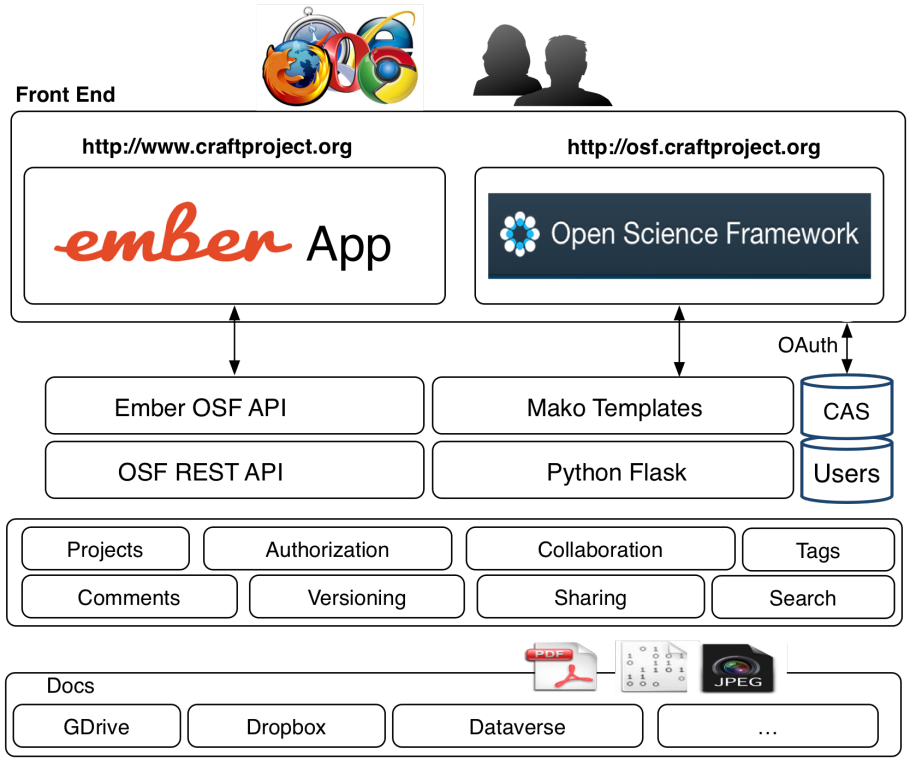


Figure 1: Overview of the CRAFT repository architecture.

OSF enables us to cleanly separate CRAFT specific development from the base system by use of their provided REST API. This is a big advantage, and allows for easier upgrades to an existing OSF installation, as new releases are released. OSF provides additional useful capabilities such as chat and WIKI surrounding project entities, versioning of files and even projects (through registration and forking). We therefore decided to capitalize on the recent advances of OSF to base the development of the CRAFT repository.

Initially, we decided to approach the development of the CRAFT repository using approach 3, whereby we attempted to recreate all the functionality required by CRAFT by developing an EmberJS application [2] using the OSF REST API. However, during our initial prototyping we realized that certain existing graphical features of OSF related to core project management, were themselves compelling to use without recreating them in our EmberJS application. The CRAFT repository uses all user, project, and file management features provided by OSF, as well as Wiki and support to add-on tools (e.g., Google Drive, Dropbox, etc.) capabilities. Further, because of DARPA requirements and privacy concerns we were unable to use the hosted OSF instance, and were required to install our own instance at a server in Notre Dame. The resulting *hybrid architecture* is illustrated below in Figure 1.

The frontend is hosted across two symbiotic servers: an EmberJS CRAFT application that has CRAFT specific capabilities such as support for chip design flows and IP cores;

and a marginally customized OSF instance [16] that we use to provide the core Project and membership capabilities. In order, to provide a consistent look and feel, we lightly customized our instance of OSF to ensure that, (a) OSF LOGO is replaced with CRAFT logo, (b) OSF name is replaced with CRAFT, and (c) menus were modified to provide link to our CRAFT EmberJS application and remove links referring to OSF functionality not used in the CRAFT repository. We have captured all our modifications in a standalone script to ensure easy upgrades of the OSF instance. We envision, that we will regularly update our OSF instance to take advantage of bug fixes and rollout of new features in the OSF framework.

We conceptualized the CRAFT architecture as two servers tying together in a seamless repository. Users see it as a single application with the same look and feel throughout the repository. This was achieved by copying the core OSF look and feel to our CRAFT EmberJS application and updating the application menu's to include links to functionalities on both the servers. OSF's bootstrap-based development integration with Semantic UI[17] components used in CRAFT application presented a major integration challenge. We summarize below, how a user interacts with the repository

- User logs in via the CRAFT application;
- The CRAFT application redirects to the Oauth CAS server to get a user token for authentication. This token is cached in the browser to satisfy both the CRAFT application and the OSF instance;
- The CRAFT application shows the user a customized dashboard composed of a set/collection of CRAFT communication channels and flows;
- When a user clicks on a project, s/he enters the CRAFT project main page;
- The CRAFT application implements two custom pages: the overview page for describing the flow or the IP cores and the *Craft* page, which integrates the view and editing capabilities for the design flow or the IP core; and
- The two EmberJS pages are included on a project menu along with the remaining links of the OSF instance. This creates a unified experience across both servers.

Our hybrid approach to architecture has allowed us to use in the repository various project-oriented features in OSF such as exposing files, revisions and releases, comments, tag, history and other components such as WIKI's. We are able to use the OSF OAuth mechanism for Centralized Authentication (CAS) without any modifications. Our development effort has mainly focused on CRAFT specific capabilities such as representing and editing chip design flows and IP cores.

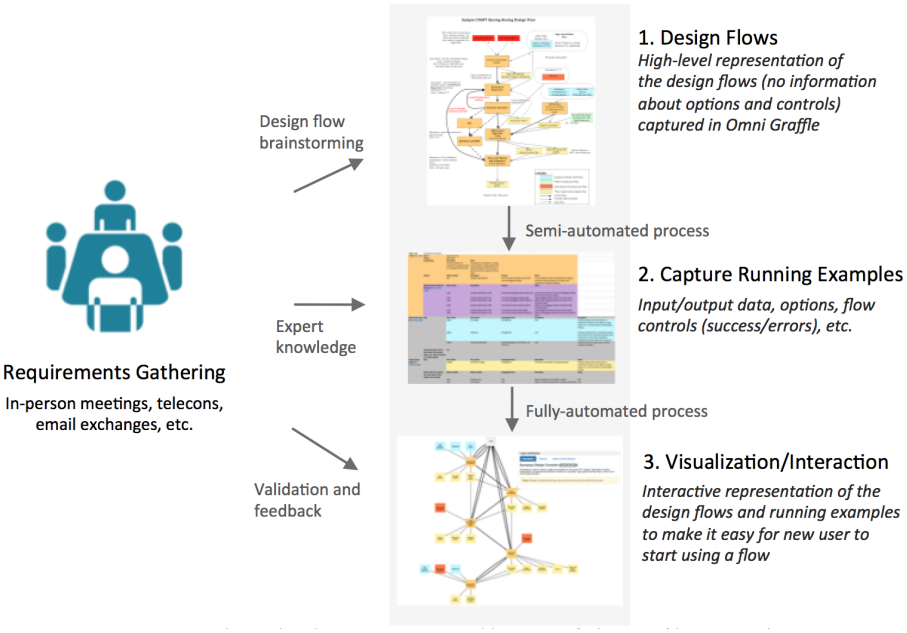


Figure 2: High level overview of the process for identifying the template used for a design flow.

4.1. CRAFT Design Flow Specification

Each CRAFT performer team is developing a user-oriented version of their new design flows that can be used by DoD designers. As such, their representations are specific to their individual design approaches. The CRAFT repository team has been tasked to be the bridge between the various flows and the target DoD design community. A major deliverable for the CRAFT repository is to capture, document, store, and visualize such flows in a systematic and common methodology. The repository flow description has to be flexible, and generic to describe the specificities of the various flows. Our initial interactions with the various teams indicated that no common design flow representation standards exits in the chip design community. Each company has their own methodology of describing their flow, very often as a text manual. In order to come up with a standard representation for representing the design flows, we took an iterative approach illustrated in Figure 2. In the first step, we conducted several requirements gathering meetings (in-person, teleconferences, email exchanges, etc.) with the performer teams to create a high-level view of their design flows (no flow specific information required at this point, e.g., options and controls). The high level views were captured described in a visual fashion using OmniGraffle [18], in a manner similar to how scientific workflows are described. The visual representation captured the steps that are involved in the flow and the input and output files required for each step. The second step involved expanding this information into a detailed description that included information about options, flow controls and data. This detailed information was captured in a spreadsheet template, that was then filled up by the various CRAFT performers. Finally, we developed a common JSON format to capture all this information. JSON was used, especially because of its ease of use with the various web based javascript visual-

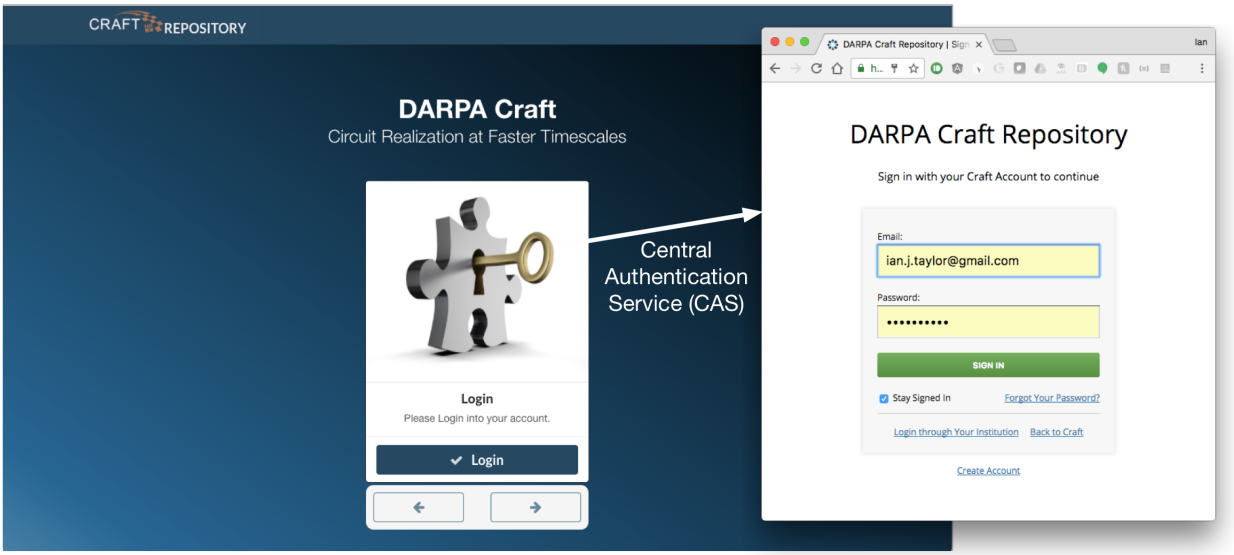


Figure 3: CRAFT login screen: authentication is performed via Oauth from OSF CAS.

ization libraries. The implementation of design flows in the repository is further described in section 6.1

5. Implementation

The CRAFT repository exposes many tools to help the CRAFT community organize information and to allow the different teams to communicate efficiently with each other. In order to develop a robust and efficient system with state-of-the-art technology, the development of the CRAFT repository is fully based on open source software. The underlying project-based implementation is provided through the OSF REST API. The Web dashboard is implemented with EmberJS and uses the Ember OSF toolkit [19] to model the REST API. Since we capitalize on OSF capabilities (provided by a private local OSF instance) for most of the GUIs and tooling (e.g., file management, etc.), the implementation of the CRAFT EmberJS application encompasses five areas of development: (1) authentication to the OSF CAS; (2) the main dashboard page; (3) the project overview Page; (4) the project publishing/registration mechanism; and (5) the discussion forum.

In addition to these specific functional elements, the EmberJS application also implements the more general look-and-feel features (e.g., menus, CSS, etc) for the application.

5.1. Authentication and Registration

A customized login screen was created for the CRAFT application, which allows a user to login or to register with the OSF, as shown in Figure 3. The registration method simply redirects the user to our private OSF instance for registration within the server. When the user logs in, authentication in the application is achieved using the Ember OSF plug in that runs the Oauth flow to the OSF CAS, which asks permission for the EmberJS application to

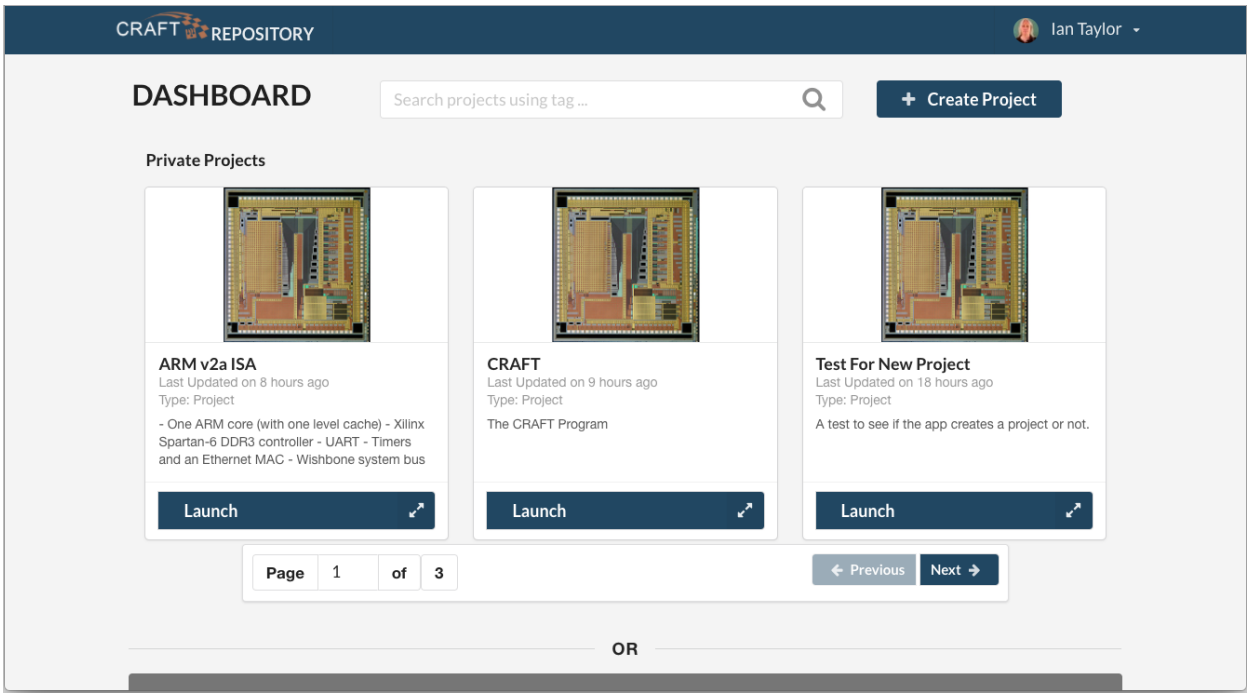


Figure 4: CRAFT main dashboard screen.

access data on the OSF (as also shown in Figure 3). Upon success, the application redirects the user to the *redirect URL* specified in this flow. The redirect URL is currently defined to be the dashboard page of the EmberJS application, i.e., sends the user to the CRAFT main dashboard.

5.2. CRAFT Dashboard

The CRAFT repository is organized into private and public projects. Private projects are projects that restrict access/management only to the members of the projects, while public projects provide view access to everyone. In the main dashboard, we organized these projects into two different category views, which explicits the nature of such projects (a screen-shot of the private project part of this dashboard is shown in Figure 4). The dashboard uses the Ember OSF pagination filtering mechanism to filter the projects to show a predefined number of projects per page. Projects navigation is done via the pagination buttons, and/or using the search mechanism—the CRAFT repository capitalizes on the tag-based search features provided by OSF. Using a combination of these features the user can locate desired projects quickly. Lastly, the dashboard allows the user to create a project and uses a modal dialog pop-up to allow the user to enter the details of the new project.

5.3. Project Overview Page

The project overview is designed in a way to provide a concise summary of the project's goals and contents. Figure 5 shows a screen-shot of a project overview page, which is composed of a project overview tab, and a list of activities tab (e.g., for auditing project usage). The project overview is separated into two panels. The upper part of the overview's

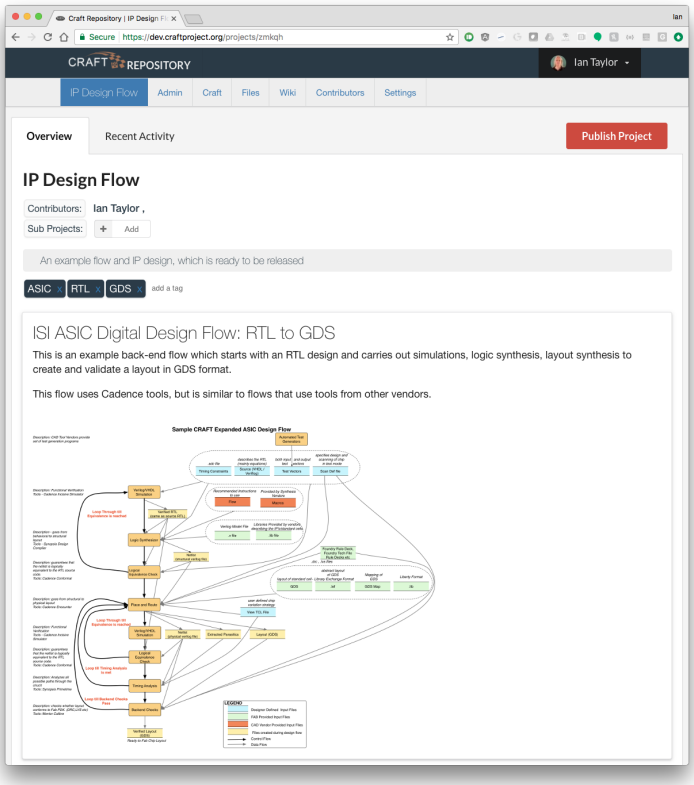


Figure 5: CRAFT project overview screen.

tab shows (1) the project title, (2) a list of contributors (members of the project), with links to their respective profiles, (3) a short description of the project, (4) a list of sub-projects (if this project has one or more sub-projects) visible only to members of the sub-projects, and (5) a list of comprehensive tags that characterizes the project (the user can also click to the right of the tags to add a new tag). On the lower part of this tab, a high-customizable project overview description is provided. This segment provides the use of the Medium Editor [20], which is a simple intuitive HTML5-based rich text editor. It provides a popup toolbar that allows text formating (bold, italic, headings, etc.), adding hyperlinks, and also dragging and dropping images from a desktop. Although, this is very simple, it provides an efficient mechanism to allow users to create a general project overview description.

5.4. Project Publishing/Registration

When a user makes a release of a project using the publish project in Figure 5, the system creates a frozen, time-stamped version of a project that cannot be edited or deleted; so while the original project can still be edited, the registered version cannot. Publishing a project allows a user to create a snapshot of the project at certain points in time or they can publish it when the project is completed and ready to share with their collaborators.

Projects can be published immediately or embargoed for up to 4 years, delaying their public release. Published project cannot be deleted, but they can be withdrawn, which

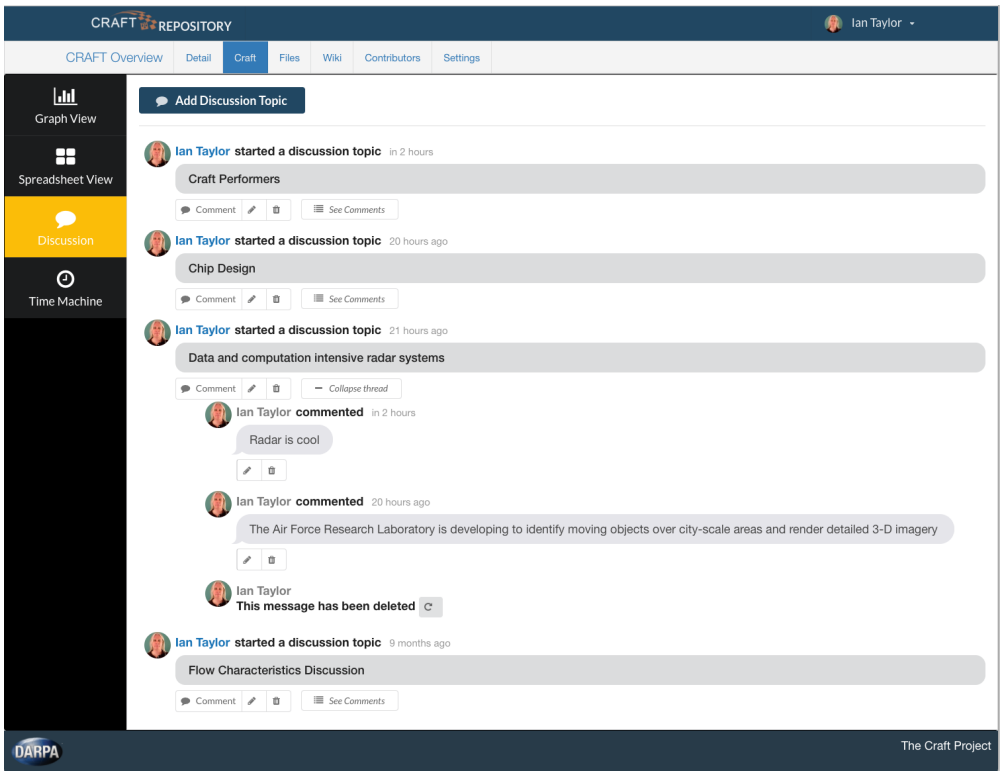


Figure 6: CRAFT discussion forum.

removes the content of the registration but leaves behind basic metadata, like published title, contributors, and an optional reason for the withdrawal.

5.5. Discussion Forum

The CRAFT repository provides a topic-based discussion forum (Figure 6), where any contributor can start a new topic or comment on an existing one. Upon a contribution is performed to any topic, the contributors is automatically subscribed to it and will be notified (via email) once any other contributor answers/replies to a comment. Discussions follow the project visibility, i.e., the discussion forum will only be public if the project is also public. For instance, a general discussion forum (e.g., DARPA announcements), can be made through a separate public project created only for this purpose. The discussion feature is built on top of OSF's chat feature, where we have re-defined the concept of simple messages into topic threads, and message replies into thread replies.

5.6. Version Control

The CRAFT repository extends the OSF built-in version control features which allows users to upload and track history of changes made to any stored files including design flows and IP cores. Uploading a file with the same name as the one stored in the CRAFT repository will automatically update the revision for that file. In addition, OSF will automatically maintain a record of all previous versions for files stored in the repository [4]. The EmberJS application has a view mode called the Time Machine (Figure 7), which enables users to

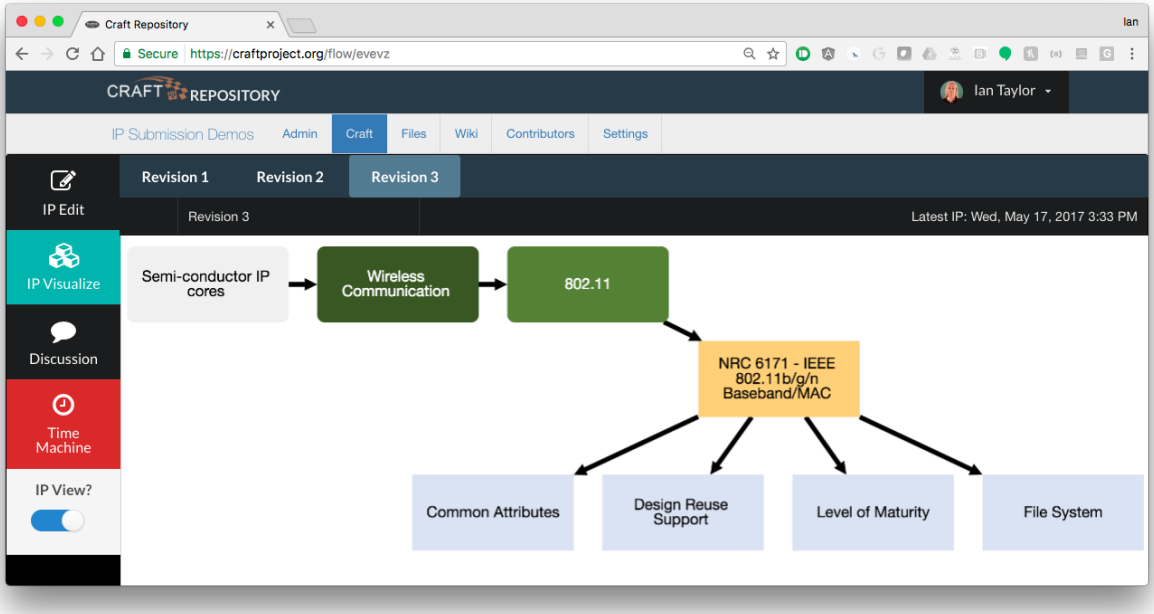


Figure 7: CRAFT Time Machine.

easily switch between the different revisions of a design flow. At any time when viewing a design flow or IP core, the user can click the Time Machine icon to bring up a list of revisions and can then select one to visualize or begin editing. Editing the flow or IP core will automatically update to a newer revision.

6. Craft Collaborative Tools for Circuit Design

This section describes our CRAFT specific extensions that we have developed to enable the specification, editing and visualization of design flows and the description of the intellectual property for each core.

6.1. CRAFT Design Flows

As part of the DARPA CRAFT program, each CRAFT performer team is developing a user-oriented version of its new design flow. A major requirement for the CRAFT repository is the ability to capture, document, store, and visualize such flows in a systematic way, where the flow representation should be flexible and generic enough to accommodate the specificities of each flow. Therefore, we have developed a common template to capture and integrate design flows from all teams, that has been revised/expanded continually as needed. The goal is to capture the steps a DoD designer will have to take to use the flow. In the first step of the process of design of this representation, we conducted several requirements gathering meetings (in-person, teleconferences, email exchanges, etc.) to create a high-level view of the design flow (no flow specific information are required at this point, e.g., options and controls). In the second step, we extracted expert knowledge to expand the high-level

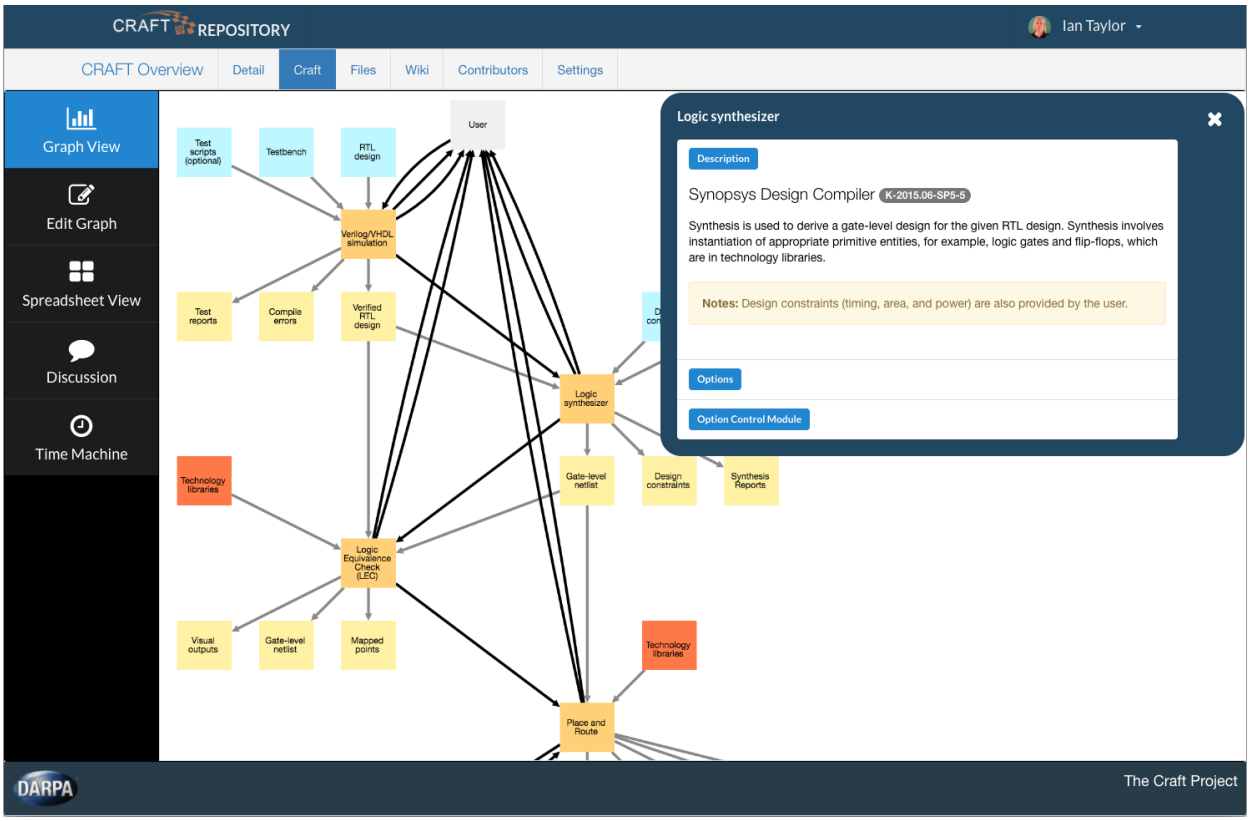


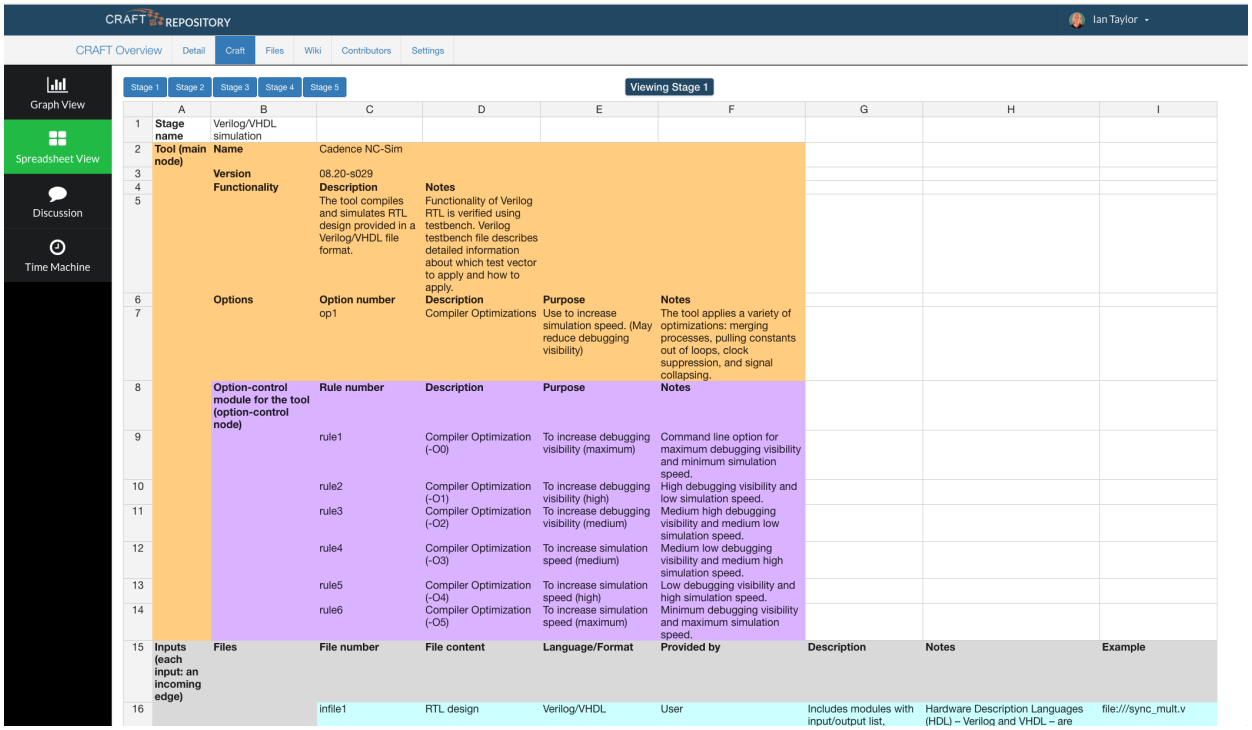
Figure 8: CRAFT design flow represented in a graph format.

flow into a complete running example flow, which included information about options, flow controls, and data. These interactions allowed us to iterate towards a common schema for documenting the flows. In the initial phase of the project, we defined the template to capture design flows as an Excel spreadsheet (facilitates visual communication with the performer teams).

One of the major capabilities that we wanted to demonstrate in this repository was the ability for the users to visualize and edit these flows, and perform flow validation (syntactically and semantically). We decided to describe and formalize the template for the design flows in JSON format[2]. Our motivation to use JSON was driven by its lightweight, compact representation and widespread use in web interfaces and modern Javascript visualization libraries for input.

Overall, the JSON schema has two major sections, *tools* and *stages*. The tools section describes a list of tools that are used within the flow and specifies the tool name and version, as well as the tool functionality and specific options or configuration parameters. The advantage of this approach is that a tool can be easily reused in stages by referencing its unique identification. The stage section describes the individual steps of the flow, and are composed of a set of input and output files (which may be provided by external vendor

[2]CRAFT Flow template schema: `https://github.com/pegasus-isi/craft/tree/master/schema`



	A	B	C	D	E	F	G	H	I
1	**Stage name**	Verilog/VHDL simulation							
2	**Tool (main node)**	**Name**	Cadence NC-Sim						
3		**Version**	08.20-s029						
4		**Functionality**	**Description**	**Notes**					
5			The tool compiles and simulates RTL design provided in a Verilog/VHDL file format.	Functionality of Verilog RTL is verified using testbench. Verilog testbench file describes detailed information about which test vector to apply and how to apply.					
6		**Options**	**Option number**	**Description**	**Purpose**	**Notes**			
7			op1	Compiler Optimizations	Use to increase simulation speed. (May reduce debugging visibility)	The tool applies a variety of optimizations: merging processes, pulling constants out of loops, clock suppression, and signal collapsing.			
8		**Option-control module for the tool (option-control node)**	**Rule number**	**Description**	**Purpose**	**Notes**			
9			rule1	Compiler Optimization (-O0)	To increase debugging visibility (maximum)	Command line option for maximum debugging visibility and minimum simulation speed.			
10			rule2	Compiler Optimization (-O1)	To increase debugging visibility (high)	High debugging visibility and low simulation speed.			
11			rule3	Compiler Optimization (-O2)	To increase debugging visibility (medium)	Medium high debugging visibility and medium low simulation speed.			
12			rule4	Compiler Optimization (-O3)	To increase simulation speed (medium)	Medium low debugging visibility and medium high simulation speed.			
13			rule5	Compiler Optimization (-O4)	To increase simulation speed (high)	Low debugging visibility and high simulation speed.			
14			rule6	Compiler Optimization (-O5)	To increase simulation speed (maximum)	Minimum debugging visibility and maximum simulation speed.			
15	**Inputs (each input: an incoming edge)**	**Files**	**File number**	**File content**	**Language/Format**	**Provided by**	**Description**	**Notes**	**Example**
16			infile1	RTL design	Verilog/VHDL	User	Includes modules with input/output list,	Hardware Description Languages (HDL) – Verilog and VHDL – are	file:///sync_mult.v

Figure 9: CRAFT design flow represented in a tabular format.

tools), a tool (referenced by the its unique identification), and flow controls, which capture the decision flow based on the tool output. Currently, the repository automatically validates the JSON file against the schema, and performs simple semantics validation such as cross-references of input/output files and tools.

We provide the users two options for describing their flows:

1. Upload a JSON document conformant to the schema. The repository UI automatically validates and identifies any errors in the document.

2. Use of an interactive visualization tool. The flow visualization and editing tool Figure 8, provides an interactive interface to visualize the flow (based on the open source Cytoscape.js project [21]). Users can click on the boxes (representing stages, input/output data) and edges (representing dependency and control flows) to visualize detailed information about each tool and its control options.

We expect the community to use the interactive visualization tool primarily for creating and describing their flows. For bulk editing of an existing design flow, we also have developed tabular visualization using the Handsontable library [22]) facilitates. This tabular viewer mimics the initial spreadsheet template, (Figure 9) of the flow.

All design flows in the repository are stored in a JSON document conformant to the schema described above. Versions (history of changes) of the flow are automatically recorded in the repository. In the repository, each project manages a single flow. We chose this approach to foster collaborative efforts, since all discussions and files within a project, are

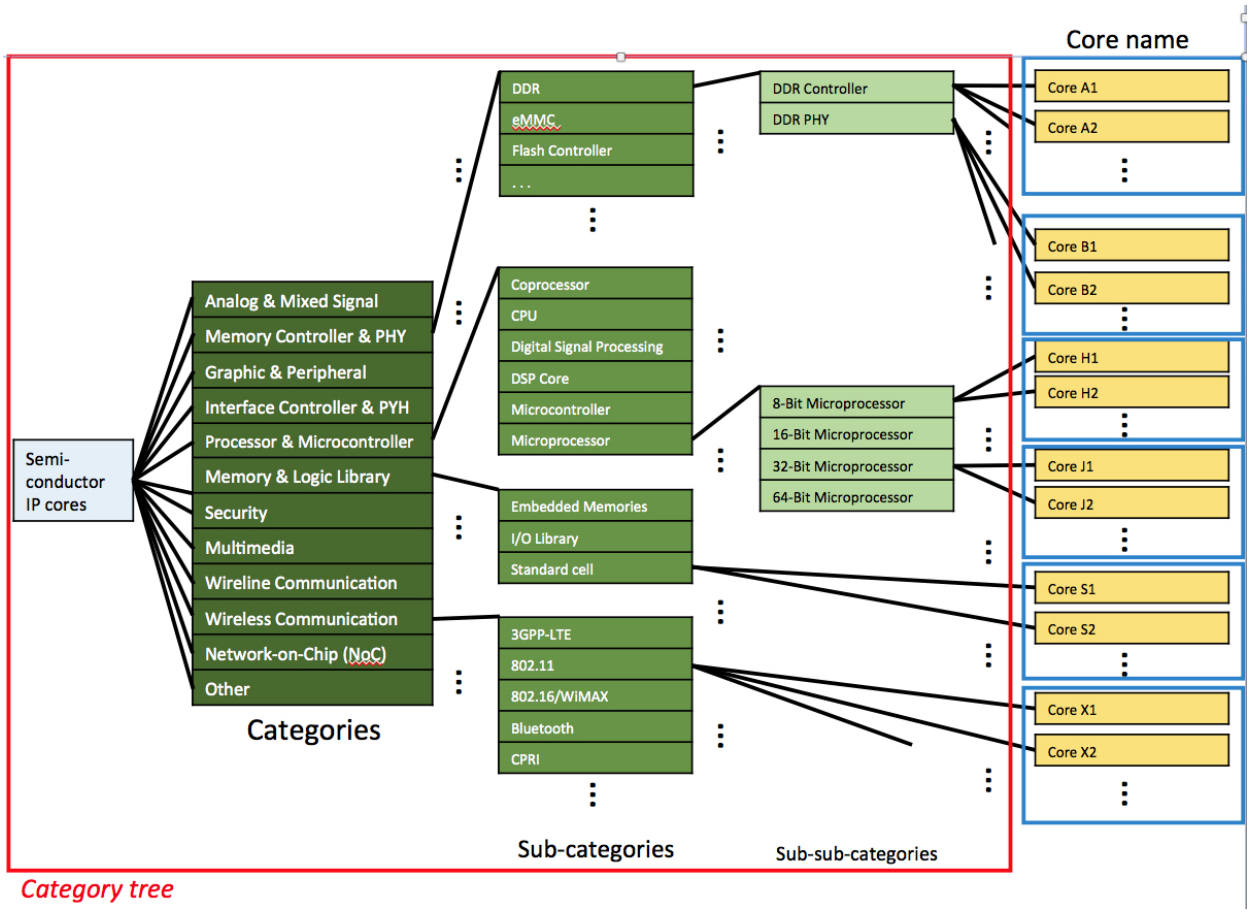


Figure 10: Sample Category based organization of IP.

tied to a single flow. Sub-projects can be used to represent multiple flows from a single performer.

6.2. CRAFT IP

Intellectual Property (also called IP cores) are pre-designed circuit modules, from libraries of small cells (e.g., logic gates) to large sub-systems (e.g., a floating-point unit, a microprocessor, etc.), that are used by ASIC designers to reduce their design effort and cost. (In turn, IP vendors profit by amortizing their cost across the large number of ASIC design projects that use their IP.) Each performer in the CRAFT program is developing multiple IP and planning to distribute these modules, along with their new tools, languages, and design flows, to the DoD ASIC design teams. Hence, our repository must provide support for CRAFT performers to submit their IP and the DoD ASIC design teams to be able to review/search the available IP, identify IP of interest, download IP they wish to use, and get support from the IP's creator and other DoD ASIC design teams who have also used the IP.

Each CRAFT performer is planning to submit a different type of IP, e.g., general purpose cell library for ASIC designs, analog to digital converters (ADC) as part of a system on chip

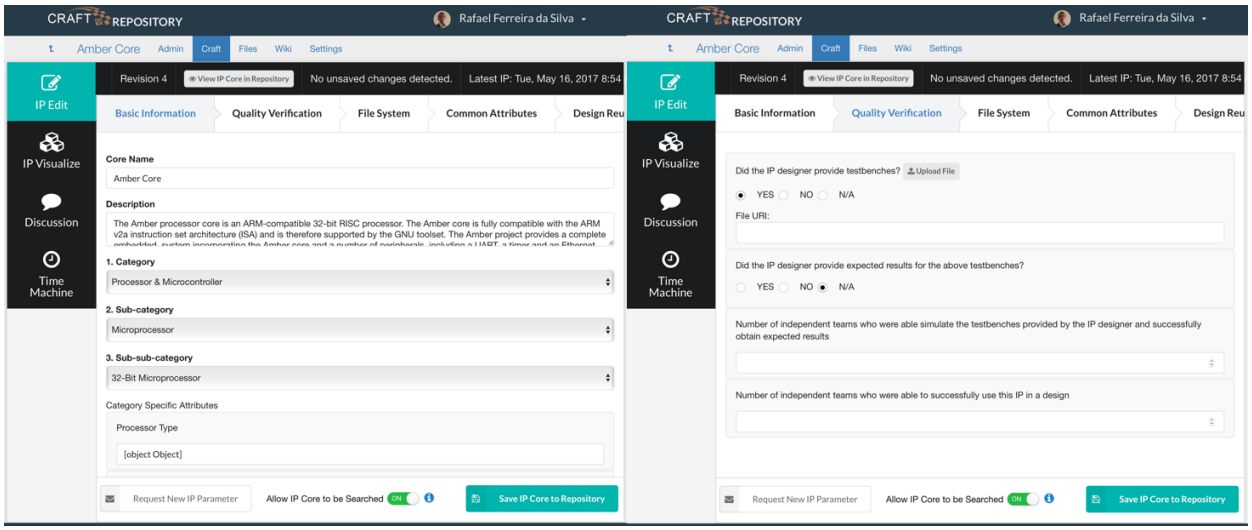


Figure 11: Intellectual design wizard for entering circuit IP, and allows category specific attributed to be entered (*left*), and quality verification attributes (*right*).

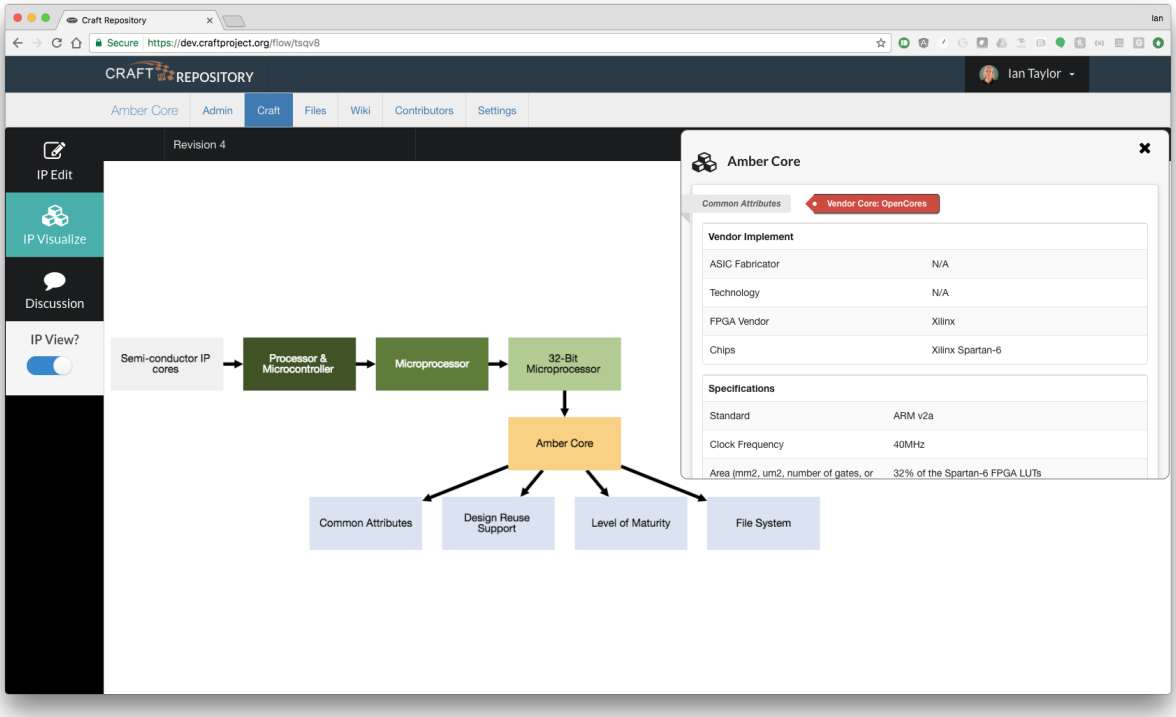


Figure 12: CRAFT IP flow visualized in a graph format.

(SOC), and high level libraries for RTL or software generators in a hardware construction library for IPs. On the basis of our study of various IPs in a popular open source hardware repository for IP cores [23], we realized that the organization of IPs should be category based (Figure 10) and hierarchal in nature. Each category can have a different set of attributes,

and categories can be organized as a category tree. Each category has category-specific attributes and also inherits attributes of its ancestors in the tree.

Based on our survey of CRAFT performers, we identified a set of common attributes that we expect each IP to inherit. These common attributes can be viewed as being associated with the single root category of the IP category tree. Similar to our implementation choice for Design flows, we decided to use JSON format for CRAFT IPs, including visualization and storage in the repository. However, as evident with the category based organization, we realized that the schema should be general purpose and extensible to allow for addition of new categories and relationships between them as well as new attributes. This is unlike our schema for design flows where we were able to upfront develop a (relatively) fixed schema and attributes.

The JSON schema for CRAFT IPs is developed using the draft-06 version[3], and utilizes the *$id* keyword and *definitions* to structure the valid set of IP categories and sub-categories. Each category (or sub-category) allowed valued is defined in an *enum*, and category-specific attributes are defined as elements of a (sub-)category. We then use the *anyOf* keyword to express the rules of inheritance for the IPs. The remaining attributes (e.g., common attributes, file system, etc.) are defined using common JSON properties.

Even though the schema described above is extensible, we deliberately chose to hide from the users the functionality to add their own categories, as we want to present a curated and consistent set of categories in the repository. Instead, we developed an IP wizard that allows users to identify the category hierarchy (Figure 11-*left*) to which their IP belongs and enter the values for its category specific attributes. There are separate tabs (Figure 11-*right*) for each of the common category that we have identified and applies to each CRAFT IP. Once the user has entered the details for the IP using the wizard, they have an option for visualizing the IP (Figure 12) as a tree structure. The visualization clearly lists the category hierarchy for the core, in this case Amber Core on the top, and the common categories as children of the core. Each category box can be clicked to bring up the list of attributes associated with that category.

7. Future Outlook

In the science domains, a way to share computational methodologies is done via the use of workflows [24, 25, 26]. Workflows describe the computational tasks that need to be executed, the data they process and generate, and the order of task execution. Many of the science workflow users have started out using scripts to coordinate computations – just as ASIC designers do today. However, this approach has limitations in how design flows can be put together and re-used. It is also error-prone and often inefficient. Workflows can help overcome these barriers. In our future work we will explore using workflows in the area of ASIC design and incorporate workflow definitions in the Craft Repository.

Workflows of design flows will capture the steps of a design flow in a discrete, structured manner to facilitate reuse, automation, parallelization and optimization. They will allow

[3] `http://json-schema.org/draft-06/schema`

users to connect various sequences of point tools and scripts using input and output files (with additional data translation where needed). Thus, other designer will be able to easily inspect the design flow, modify individual components, or change the component parameters. In addition to the reusability of entire flows, it will also be possible to reuse sub-designs, incorporating previously characterized and verified components to significantly reduce the time to completion of an ASIC design.

We also plan to use workflow technologies to perform quality control on data being submitted to the CRAFT repository. Basing our formats in JSON will allow us to vary these if they are uploaded as files. We can also use workflows to run specific checks on the data being deposited int the repository, for example checking the data types and the data value ranges.

8. Conclusion

This paper presented the CRAFT repository, a collaborative science gateway for accelerating circuit realization. The main goal of the repository is to capture and document, in a systematic way, the design flow process for chip design development. The repository is built using a hybrid approach, where the front-end has been developed as an EmberJS application, which interacts with an instance of the Open Science Framework. The application was customized to fulfill CRAFT program requirements, in particular the development of templates and schema to capture the design flows. Currently, the repository is only available to CRAFT performers but will be soon made available to wider community of DoD ASIC designers. CRAFT performers include: Carnegie Mellon University, Harvard University, Princeton University, Stanford University, University of California Berkeley, University of California, San Diego, and the University of Southern California. Industry collaborators and government organizations include the Boeing Company, Cadence Design Systems, Inc., DARPA, NVIDIA Corporation, Northrop Grumman Corporation, and Synopsys, Inc. Future work include the development of a service to capture and document intellectual property (IP) cores. Usability studies will follow.

9. Acknowledgements

This work was funded by DARPA under contract #HR0011-16-C-0043 “Repository and Workflows for Accelerating Circuit Realization (RACE)”.

References

[1] Circuit Realization at Faster Timescales (CRAFT), `http://www.darpa.mil/program/circuit-realization-at-faster-timescales`.

[2] The Craft Repository, `https://craftproject.org/`.

[3] EmberJS, `http://emberjs.com/`.

[4] The Open Science Framework, `http://www.osf.io/`.

[5] I. J. Taylor, A. Brinckman, E. Deelman, R. Ferreira da Silva, S. Gupta, J. Nabrzyski, S. Park, K. Vahi, Accelerating circuit realization via a collaborative gateway of innovations, IWSG 2017: 9th International Workshop on Science Gateways accepted.

[6] Lynx design system, https://www.synopsys.com/implementation-and-signoff/lynx-design-system.html.
[7] The Bitbucket Website, http://bitbucket.org/.
[8] The Github Website, http://github.com/.
[9] J. R. Spies, The open science framework: improving science by making it open and accessible, University of Virginia, 2013.
[10] Dropbox, https://www.dropbox.com.
[11] Google Drive, https://www.google.com/drive/.
[12] Box, https://www.box.com/.
[13] JSON-API Specification, http://jsonapi.org/.
[14] Django Rest Framework, http://www.django-rest-framework.org/.
[15] M. McLennan, R. Kennell, Hubzero: a platform for dissemination and collaboration in computational science and engineering, Computing in Science & Engineering 12 (2).
[16] Craft OSF Repository, https://osf.craftproject.org/.
[17] Semantic UI, http://semantic-ui.com/.
[18] Omnigraffle: Diagramming and graphic design tool for mac, https://www.omnigroup.com/omnigraffle.
[19] The Ember OSF Toolkit, https://github.com/samchrisinger/ember-osf.
[20] HTML5 Medium Editor, https://github.com/yabwe/medium-editor.
[21] Cytoscape.js, http://js.cytoscape.org/.
[22] Handsontable – javascript spreadsheet, https://handsontable.com/.
[23] Design reuse: Design and reuse, the system-on-chip design resource - ip, core, soc, https://www.design-reuse.com.
[24] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, J. Vetter, The future of scientific workflows, The International Journal of High Performance Computing Applications accepted. doi:10.1177/1094342017704893.
[25] E. Deelman, K. Vahi, M. Rynge, G. Juve, R. Mayani, R. Ferreira da Silva, Pegasus in the cloud: Science automation through workflow technologies, IEEE Internet Computing 20 (1) (2016) 70–76. doi:10.1109/MIC.2016.15.
[26] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, E. Deelman, A characterization of workflow management systems for extreme-scale applications, Future Generation Computer Systems accepted. doi:10.1016/j.future.2017.02.026.