Accelerating Circuit Realization via a Collaborative Gateway of Innovations

Ian J. Taylor[†*], Adam Brinckman[†], Ewa Deelman[‡], Rafael Ferreira da Silva[‡] Sandeep Gupta[§]
Jarek Nabrzyski[†], Soowang Park[§], and Karan Vahi[‡]
[†]Center for Research Computing, University of Notre Dame, Notre Dame, IN, USA
[*]School of Computer Science & Informatics, Cardiff University, Cardiff, UK
[‡]Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA
[§]Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
Emails: {itaylor1, abrinckm, naber}@nd.edu, {deelman, rafsilva, vahi}@isi.edu, {sandeep, soowangp}@usc.edu

***Abstract*—The CRAFT repository seeks to develop new fast-track circuit-design methods, multiple sources for integrated circuit fabrication, and a technology repository. This paper presents the design decisions and implementation to build a collaborative repository that capitalizes on the recent advances in open-source collaborative frameworks, and answers the needs for the DARPA's CRAFT program. The repository has been developed as an EmberJS application (front-end), which interacts with an instance of the Open Science Framework (OSF).**

***Keywords*—Collaborative Environments; Design Flows; Chip Design**

I. Introduction

The Circuit Realization at Faster Timescales (CRAFT) DARPA (Defense Advanced Research Projects Agency) program [1] aims at reducing the design cycle time needed for creating custom integrated circuits. Currently, the design of a custom integrated circuit for a specific task can take more than two years, require a large team of engineers, and can cost up to $100 million. Such timescales and economics are not practical and therefore DoD (Department of Defense) engineers instead use readily available inexpensive general-purpose circuits and implement the specialized operations using software. 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 chips (or ASICS—application-specific integrated circuits) 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. Our CRAFT specific additions enhance the overall chip design process by facilitating re-use of circuit modules and rapid adoption of new design flows across CRAFT project participants.

II. 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 for ASIC designs. The community of researchers and industry experts working to address this challenge will develop completely new tools, new types of libraries, and new design flows (a design flow is a sequence of steps, each carried out by a tool; we describe it in more detail later on). Second, the program requires CRAFT researchers to transfer immediately all their new tools, libraries, and 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 should 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 and the reposi-

tory's flexible architecture to create different permissions for different projects and topics enables this. The following five sections discuss the underlying requirements and the high level concepts we use to attain CRAFT goals.

A. 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 online tools, such as Bitbucket [5] and GitHub [6]. 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 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 lists 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 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.

B. 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.

C. 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.

D. 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.

E. 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.

III. Architecture and Design

In order to produce a repository that met the requirements of the CRAFT community, architectural design decisions needed to carefully taken to optimize development and enable sustainability. We then were faced with different approaches that would allow 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.

Although option 1 is generally the simplest approach, as there are no other systems to learn and integrate with, and creating an architecture with such clear requirements is a straight forward engineering task, this approach has major disadvantages. First, the timescale we had to work within was around 7 months, from the project requirements gathering to a first production release of the repository, which would have been a challenging target using our allocated 1.5 full time developers. Second, many of these features already exist on other websites; e.g., Github and Bitbucket already have many of the project-oriented features implemented, amongst a number of others. Third, sustainability is not guaranteed, and would require on-going additional effort to maintain the code (which would be a poor practice for reinventing the wheel). Given the limited timescale, we aimed to assess several external well-established systems against our needs. The results of this study led to two candidate systems which were investigated in depth.

HUBzero [7] is an open source software platform for building websites that support scientific activities. The signature service of a hub is its ability to deliver interactive graphical simulation tools; you can zoom in on a graph, rotate a molecule, probe isosurfaces of a 3D volume—interactively, without having to wait for a web page to refresh. Each hub is a place for users to come together and share information. The sharing mechanism enables users to create and manage their own groups of users. Any registered user can create a group and invite others to join it. The creator can accept or reject group members, and can promote various members to help manage the group. Resources associated with a group can be kept private, meaning that their access is limited to members of the group. You can also share files and each hub supports the creation of "topic" pages, which are similar to the Google "knol" model for knowledge articles. HUBzero is written in PHP and supports most of the CRAFT requirements. However, it only supports option 2 above, and the CRAFT repository would have to be integrated into the HUBzero code base. Although there are good interfaces to support this development, there is no comprehensive Web API that can be used to separate the HUBzero instance from the Web dashboard.

The Open Science Framework (OSF) [4], [8] is an open source software project that facilitates open collaboration in science research and has a main focus on the reproducibility of research. However, more recently it has evolved into a more general framework for enabling core project-based functionality through an OSF REST API [9]. Research collaboration is built around the concept of a project (called a node or component in OSF) and a user can invite other people to collaborate on a project and assign them administrator, read, or read/write permissions. It has a number of tools to facilitate import and exporting data into the repository, and it also interfaces with several popular cloud-based storage systems, e.g., Dropbox [10], Google Drive [11], and Box [12]. The file system also supports versioning. This allows a user to create new versions of a file by re-uploading the latest copy without changing the name. Each file has a set of revisions that are accessible. The OSF includes forking and registration of projects that enables us to make immutable releases. It also includes preprints for publishing a paper (or documentation) about a project. Within a project, there are multiple tools for visualizing files, for adding comments and it includes a full WIKI capability, for integrating documentation. OSF supports almost all of the underlying features that CRAFT needs, and the REST API supports all of the functionality through a JSON-API [13] compatible REST interface. The OSF is written in python, it uses the Django Rest Framework [14] to host the API, and provides an EmberJS [3] toolkit that allows third party implementations to reuse the models of the OSF API from a Javascript application.

In the context of the CRAFT project, OSF presented several advantages over HUBzero. The full access to the OSF functionality via REST API allows a complete and clear separation of the development of CRAFT-related tools from the base system. It also has far more comprehensive support for projects, includes collaborative tools, such as chat surrounding all project entities; and also provides 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 explored option 3 above by building the repository entirely in EmberJS, and by recreating the tools we needed into our GUI. However, after prototyping the initial dashboard we found several graphical features of the OSF Website itself compelling to reuse, rather than rewrite. Further, since DARPA had a requirement that all data to be hosted at a private server in Notre Dame, we were required to deploy our own instance of the OSF. This resulted in a *hybrid architecture*, which is shown in Figure 1.

The architecture for the CRAFT repository therefore consists of two symbiotic servers: an EmberJS CRAFT application [2], which contains the custom additions needed for the CRAFT project, and a marginally customized OSF instance [15], which is a re-skinned OSF instance with minor modifications. The modifications to the instance were architected to be captured in a script and just change three aspects of the OSF repository: the LOGO, the OSF name is changed to CRAFT; and the menus are modified to include the EmberJS application links and to remove some of the OSF functionality

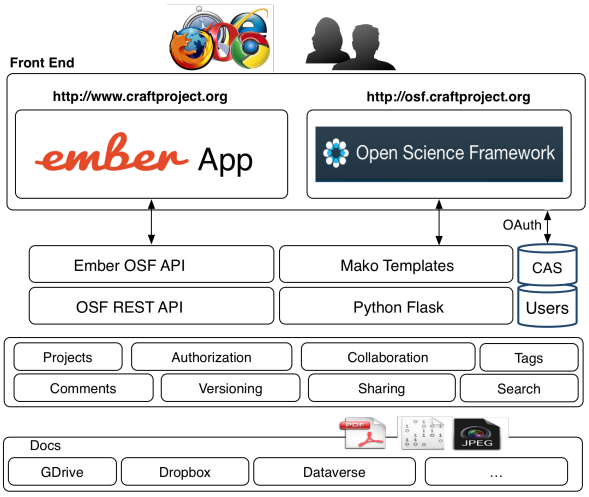

Fig. 1. Overview of the CRAFT repository architecture.

that are not required. This minor overlay allows us to easily upgrade our OSF instance and take advantage of any new additional features and bug fixes that are included in future releases of the OSF framework.

The CRAFT architecture was conceptualized as tying the two servers together into a seamless repository: the application menus contain links to functionalities hosted on both servers; and the CRAFT EmberJS application would copy the look and feel of the OSF repository, so that it appears as a single application to the users. A major challenge of this integration was to accommodate OSF's bootstrap-based development, with Semantic UI [16] components used in CRAFT application. The resulting user experience from such a design is summarized as follows:

- 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 and the *Craft* page, which integrates the view and editing capabilities for the design flow;
- 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.

Using this approach, CRAFT can take advantage of the various project-oriented features in the OSF instance to expose files, revisions and releases, tags, history, comments and other components (e.g., WIKIs) associated with the project. It can also use the OSF Oauth mechanism without modification to enable centralized authentication (CAS). All implementation for CRAFT therefore is focused on the CRAFT-specific, data, metadata, and project-specific ontologies required by the CRAFT program.

A. CRAFT Design Flow Specification

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—which 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 conduct several requirements gathering meetings (in-person, telecons, 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 extract expert knowledge to expand the high-level flow into a complete running example flow, which includes information about options, flow controls, and data. The final step, is to publish the flow into the repository, where an interactive visualization tool allows other designers to explore detailed information about the flows.

IV. Implementation

The underlying project-based implementation is provided through the OSF REST API. Our Web dashboard is implemented using EmberJs and uses the Ember OSF toolkit [17] to model the REST API. Since we capitalize on OSF capabilities (provided by our OSF instance) for most of the GUIs and tooling (e.g., file management, etc.), the implementation of the CRAFT EmberJS application is focused on five areas of development:

1) Authentication to the OSF CAS;
2) The main dashboard page;
3) The project overview Page;
4) The Craft Design Flows tooling; 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.

A. Authentication

A customized login screen was created for the CRAFT application, as shown in Figure 2. This login screen allows a user to login or the user can rotate the "shape" widget to register with the OSF. The registration method simply redirects the user to the OSF instance for registration with the server. When the user logs in, authentication in the application is achieved using the Ember OSF plug in which runs the Oauth flow to the OSF CAS, which redirects the user to the *redirect URL* specified in this flow. The redirect URL is defined to be the dashboard page of the EmberJS application, which sends the user to the CRAFT main dashboard.

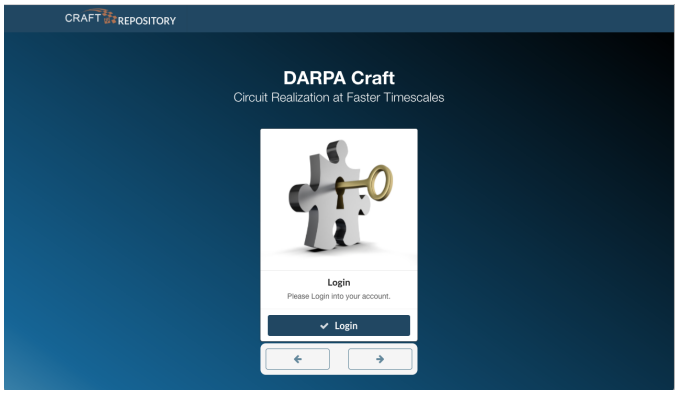

Fig. 2. CRAFT login screen: authentication is performed via Oauth from OSF CAS.

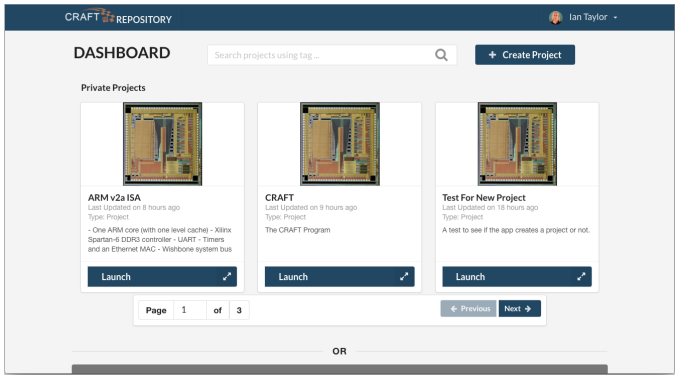

Fig. 3. CRAFT main dashboard screen.

B. CRAFT Dashboard

The main dashboard shows the list of private and public projects. A screen-shot of the private project part of this dashboard is shown in Figure 3. Private projects are projects that restrict access only to the members of those projects and public projects provide view access to everyone. The dashboard makes use of 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 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.

C. Project Overview Page

Figure 4 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 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 a sub-project) visible only to members of the sub-projects, and (5) a list of comprehensive tags that characterizes the project (the user

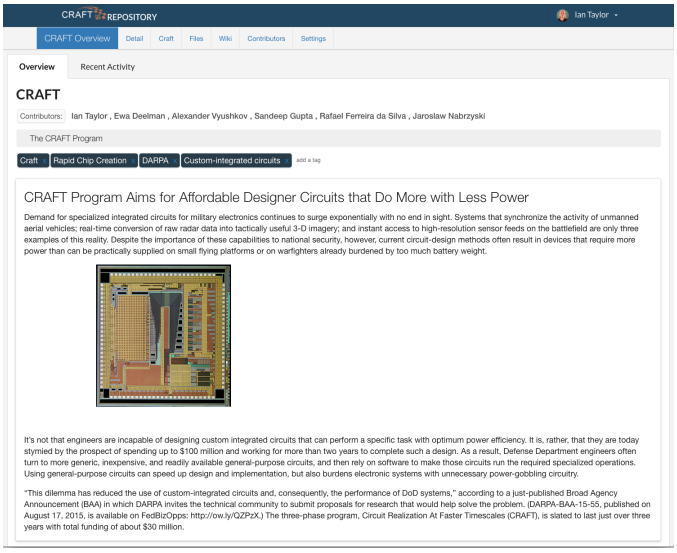

Fig. 4. CRAFT project overview screen.

can also click to the right of the tags to add a new tag). On the lower part of this tab, a project overview description is provided. This segment employs the use of the Medium Editor [18], which is a simple intuitive HTML5-based 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.

D. CRAFT Design 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). However, in order to enable flow validation (syntactically and semantically), and automated visualization/edition, subsequently flows are described in a JSON format[1]. This format is only used for internal purposes, i.e., to store the flow into the repository. Performers only interact with the visualization graph tool (Figure 5) or a tabular viewer (mimics the initial spreadsheet template, Figure 6) of the flow.

Flow visualizations are automatically generated from the JSON file. The graph visualization tool provides an interactive interface to visualize the flow (based on the open source Cytoscape.js project [19]). 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. A tabular visualization (develop with the Handsontable library [20]) facilitates to visualization/edition of detailed information at once. 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 tied to a

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

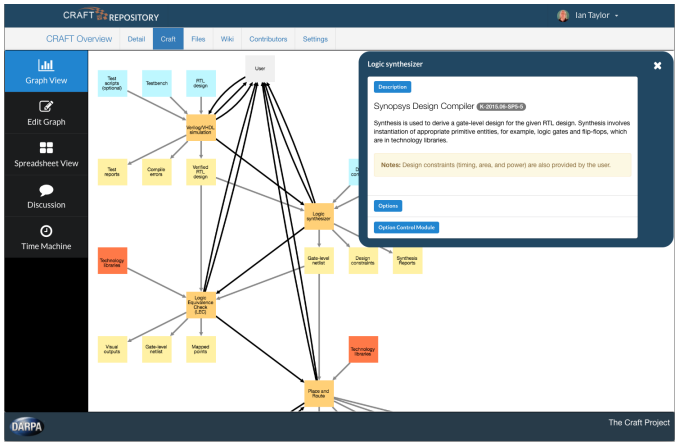

Fig. 5. CRAFT design flow represented in a graph format.

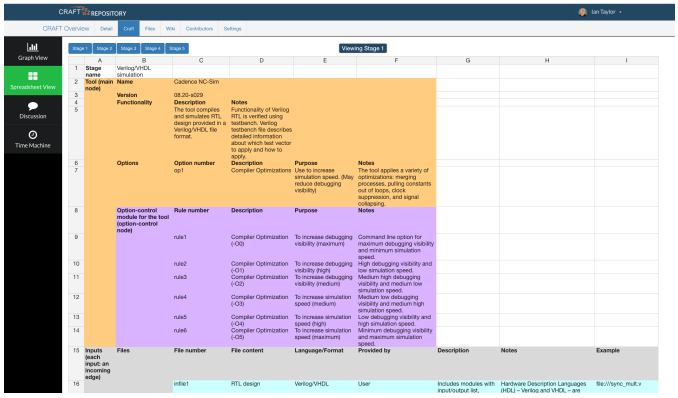

Fig. 6. CRAFT design flow represented in a tabular format.

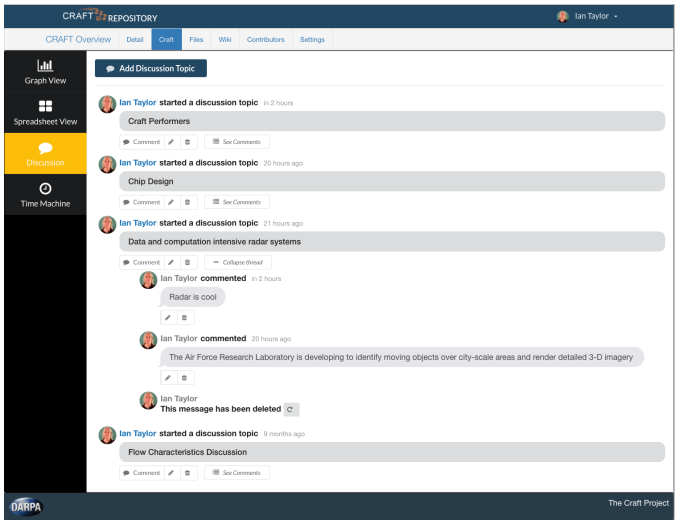

Fig. 7. CRAFT discussion forum.

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

E. Discussion Forum

The CRAFT repository provides a topic-based discussion forum (Figure 7), where any contributor can start or comment a new topic. Contributors that have interacted with a topic, will be notified (via email) once any other contributor answers/replies 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.

V. 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 made available to wider DoD community soon. 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.

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] "The Bitbucket Website," http://bitbucket.org/.
[6] "The Github Website," http://github.com/.
[7] M. McLennan and R. Kennell, "Hubzero: a platform for dissemination and collaboration in computational science and engineering," *Computing in Science & Engineering*, vol. 12, no. 2, 2010.
[8] J. R. Spies, *The open science framework: improving science by making it open and accessible*. University of Virginia, 2013.
[9] "The Open Science Framework REST API," https://api.osf.io/v2/.
[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] "Craft OSF Repository," https://osf.craftproject.org/.
[16] "Semantic UI," http://semantic-ui.com/.
[17] "The Ember OSF Toolkit," https://github.com/samchrisinger/ember-osf.
[18] "HTML5 Medium Editor," https://github.com/yabwe/medium-editor.
[19] "Cytoscape.js," http://js.cytoscape.org/.
[20] "Handsontable – javascript spreadsheet," https://handsontable.com/.