Polymath: A Platform for Rapid Application Development of Modular EDA Tools

Taimur Rabuske
INESC-ID, Lisboa, Portugal - Email: taimur.rabuske@ieee.org

***Abstract*—It is imperative that new solutions in Electronic Design Automation (EDA) appear to cope with the increasing complexity of modern chips. However, the lack of broadly available platforms for rapid application development (RAD) of EDA tools hinders the volume and quality of the contributions from the scientific community. In this paper, we tackle this issue by proposing a RAD platform for EDA tools that enables the contributors to focus on the problem that they want to solve instead of "reinventing the wheel". The proposed platform encompasses a user-friendly Tcl shell, a standardized data model, templates for quick creation of commands, a system-level Qt Graphical User Interface (GUI) and a user customizable Tk GUI, all synchronized by an event loop orchestrator with distributed processing capabilities. The amount of "boilerplate" code is reduced to a minimum in each stage of development. Finally, we propose the usage of a Continuous Integration/Continuous Deployment cycle to reduce the efforts on distribution of the tools developed on top of the platform. The platform was validated with the development of modules for the design flow of mixed-signal circuits.**

***Index Terms*—RAD, rapid application development, EDA, eletronic design automation, CAD, computer-aided design**

I. Introduction

A variety of Electronic Design Automation (EDA) tools has been devised in the past decades to cope with the increasing design complexity of electronic circuits. These tools are used to resolve diverse problems in different domains of electronic design, such as system-level, integrated and board design. The EDA industry, being an integral part of the semiconductor industry, was an essential player and enabler of a succession of technological revolutions, including the "Internet Revolution" in the 90s and, more recently, the unprecedented growth of the mobile communication. The EDA industry still undergoes consistent growth, with more than 6 billions USD in revenue in 2018, pushed today by the booming automotive and artificial intelligence sectors [1]. It is expected that the market will value more than 9 billion USD by 2024, following the broader adoption of Internet-of-Things (IoT) solutions. In fact, it is believed that reaching the IoT Tera-scale (trillions of devices deployed worldwide) will be possible only because the EDA industry will provide solutions that act as "workforce multipliers", to confront the limitations in specialized human resources needed to meet this burdensome goal [2].

Let us take a look into the anatomy of a modern EDA tool, which at its core has the functionality to solve a specific problem in the design of electronics, such as logic or physical synthesis, simulation, verification, analog design, signoff or others. This functionality is accessed generally by a Graphical User Interface (GUI), while most solutions also expose their core functions through an Application Programming Interface (API) or a scripting language, enabling the users to reuse or even reprogram its procedures to meet their needs more efficiently. An API may be accessible by an external programming language, while a scripting language is generally embedded within the application itself. In the latter case, most applications employ Tcl (Tool Command Language), which has its roots in the early 80s, and was initially purposed exactly for integrated circuit (IC) design [3], while some developers opt for a proprietary language, such as the LISP-like SKILL [4]. The presence of an embedded language means that some sort of read-eval-print loop (REPL), or "shell" must be available to the user. Finally, a problem-dependent data model exists within the applications, where the data that is relevant to the problem to be solved is stored and managed. It is very common for layout-related applications to employ the OpenAccess [5] database, which is the outcome of community effort to provide standardized data structures for interoperability between tools from different vendors. Other applications may use different data types such as netlists, hardware description language (HDL) or custom tree data structures.

While the implementations vary broadly, these key elements (core functionality, GUI, shell, data model) are found in most academic and commercial EDA tools. Though conceptually simple, a significant effort is required to build up the minimal platform that comprises these fundamental aspects that enable a competitive modern EDA tool. The lack of a widely-available standard platform for rapid development of EDA tools poses a heavy burden on developers that want to unravel aspects of electronics design: they are unable to focus entirely on the problem that they are willing to solve, and need to "reinvent the wheel" if they want their solution to be widely available to other users. Alternatively, they may end up with a sub-optimal solution in terms of user-experience, hindering the adoption of the proposed solution. From a corporate perspective, the same lack of a common platform may increase the time-to-market, which in turn has impact on sales and early-adoption.

In this paper, we present "Polymath", a rapid application development (RAD) platform devised for modern EDA tools. The proposed solution is modular, in order to be easily extensible and allows for easy integration of source code in multiple programming languages, e.g. C/C++, Python, Tcl. The presented platform minimizes the amount of "boilerplate" code that is required to provide a satisfactory solution in terms of user-experience, and provides the mentioned main aspects of a competitive EDA solution and the gluing logic that allows these bits to work together. In the next section, we present the design philosophy of the platform, while describing its

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

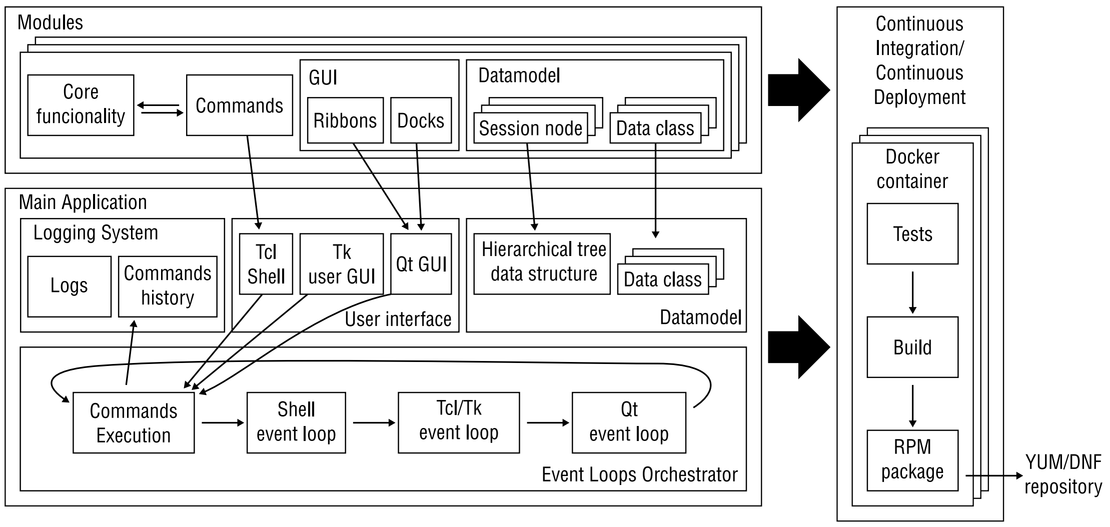


Fig. 1. Overview of the Polymath RAD platform for EDA tools.

main features and implementation details. Next, in Section III, we propose a Continuous Integration/Deployment cycle that allows fast and reliable deployment of applications built upon Polymath. Finally, Section IV discusses and concludes this work.

II. RAD PLATFORM FOR EDA TOOLS

An overview of the Polymath RAD platform is shown in Fig. 1. The system comprises a main application which, by itself, does not provide any EDA functionality. The main application allows modules to be registered at runtime, which provide all the "core" functionality to undertake a given EDA problem, which are exposed through commands that can be called in the shell or by the GUI. The modules also provide GUI elements (a "ribbon" menu and dock-able widgets) and data structures to be registered in the application data model. These components and the main features of Polymath are described below.

A. *Design Philosophy*

Similar to other EDA tools, all functionality in Polymath comes in the form of "commands". Enforcing this philosophy has a few beneficial outcomes. First, these commands are registered into the Tcl shell, and become callable functions within the application. Thus, any procedure devised by the user through a sequence of commands can be turned into a "script", enabling a higher degree of automation and repeatability. Second, it becomes easier to integrate multiple programming languages into the environment, because a command can be thought of as a "wrapper" for a function (Polymath accepts C/C++, Python and Tcl out-of-the-box but other languages can be easily integrated through SWIG [6] wrappers). Finally, when all functionality comes in the form of commands, it becomes easier to write automated tests for the application. This philosophy implies that most of the user interaction through the GUI should be converted into command calls before executed. From the perspective of repeatability, this is also beneficial because the steps achieved through the GUI may be converted into a script, i.e. it is more efficient to "source" a Tcl script than to achieve the same functionality through point-and-click interactions.

B. *Tcl Shell and Commands Creation*

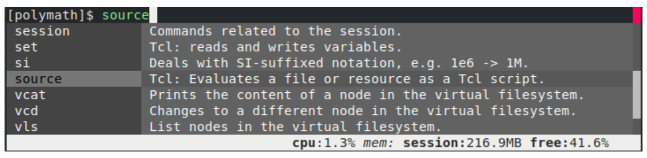
The Tcl commands registered by the modules may be used in control loops (e.g. *if*, *for* and *while*) and all other Tcl language constructs and are accessible by a REPL interface which provides syntax highlight and context-sensitive auto-completion, as depicted in Fig. 2. The registration of new commands is simple and the platform provides shortcuts for self-documentation and command-line options/arguments. Consider the following Python code excerpt that registers a dummy command "plot-data", which simply plots a sinusoidal wave with a given frequency and amplitude (options) and name (argument) into a widget.

```python
@command()
@option("--frequency", default=100e3,
  help="Frequency of the sinusoid to be plotted.")
@option("--amplitude", default=0.95,
  help="Amplitude of the generated sinusoid.")
@argument("name", type=str)
def plot_data(frequency, amplitude, name):
  """Plots dummy data."""
  y = amplitude * sin(frequency * 2 * pi * x )
  info(f"Plotting sine {name} with {frequency} Hz.")
  set_plotdata((x, y, 1))
```

Each registered command has an automatically-generated "-h" option which shows its documentation. Fig. 3 displays the results of "plot-data -h" and the auto-completion for this command. Also, the application maintains a commands history and a log files, "polymath.cmd" and "polymath.log", respectively, where the user may revisit the previously executed commands and their corresponding results and logging level (debug, info, warning and error).

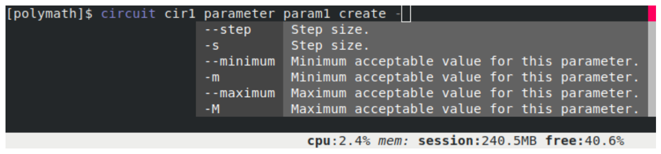
C. *Graphical User Interface*

The GUI of a Polymath-driven application consists of two parts: a Qt-based [7] system GUI and a Tk-based [8] user GUI. The Qt GUI is populated by the modules through



```
[polymath]$ source
 session          Commands related to the session.
 set              Tcl: reads and writes variables.
 si               Deals with SI-suffixed notation, e.g. 1e6 -> 1M.
 source           Tcl: Evaluates a file or resource as a Tcl script.
 vcat             Prints the content of a node in the virtual filesystem.
 vcd              Changes to a different node in the virtual filesystem.
 vls              List nodes in the virtual filesystem.
                             cpu:1.3% mem: session:216.9MB free:41.6%
```

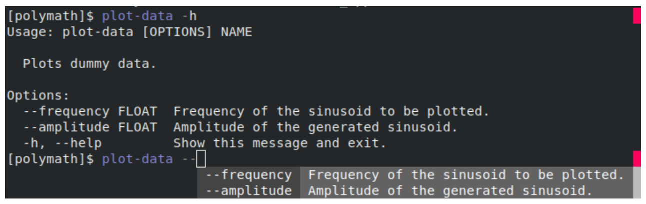
(a) Auto-completion of commands.



```
[polymath]$ circuit cir1 parameter param1 create -
                        --step     Step size.
                        -s         Step size.
                        --minimum  Minimum acceptable value for this parameter.
                        -m         Minimum acceptable value for this parameter.
                        --maximum  Maximum acceptable value for this parameter.
                        -M         Maximum acceptable value for this parameter.
                                cpu:2.4% mem: session:240.5MB free:40.6%
```

(b) Auto-completion of options from a command.

Fig. 2. Tcl shell showing syntax highlighting and context-sensitive auto-completion.

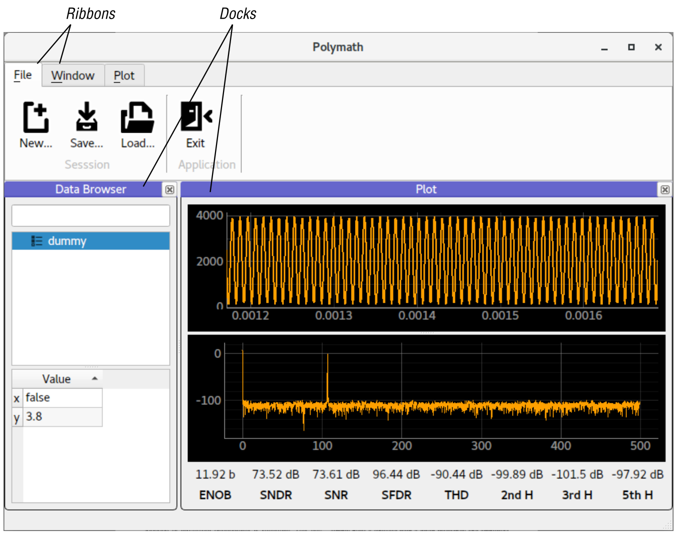


```
[polymath]$ plot-data -h
Usage: plot-data [OPTIONS] NAME

  Plots dummy data.

Options:
  --frequency FLOAT  Frequency of the sinusoid to be plotted.
  --amplitude FLOAT  Amplitude of the generated sinusoid.
  -h, --help         Show this message and exit.
[polymath]$ plot-data --
                        --frequency  Frequency of the sinusoid to be plotted.
                        --amplitude  Amplitude of the generated sinusoid.
```

Fig. 3. Tcl shell showing auto-generated commands documentation.

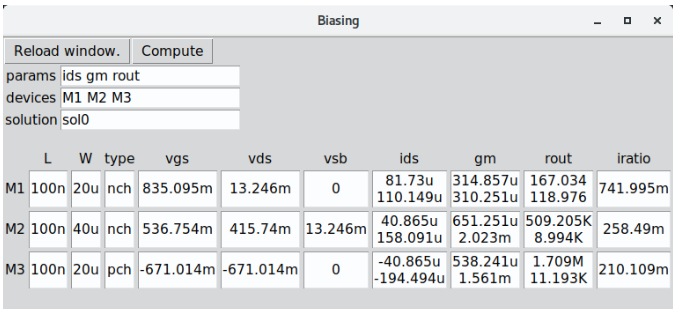
ribbons and dock-able widgets, as shown in Fig. 4. While, the main application provides only a few basic ribbons and docks for navigating the tree data model and the help system, showing/hiding docks and saving/loading the session state and GUI configuration, the actual EDA GUI functionality is provided by the modules.

The Tk-based GUI can be used by the user to extend the functionality of the application via custom windows at runtime. The Tk widgets, such as a push button or a text input box, can be linked to any Tcl commands, including commands registered by any Polymath module, facilitating the user experience. Take the following example excerpt of a Tcl code that extends the functionality of a Polymath-driven module for the sizing of analog circuits.



11.92 b	73.52 dB	73.61 dB	96.44 dB	-90.44 dB	-99.89 dB	-101.5 dB	-97.92 dB
ENOB	SNDR	SNR	SFDR	THD	2nd H	3rd H	5th H

Fig. 4. Qt-based GUI showing ribbons and docks.



Biasing

Reload window. Compute

params ids gm rout
devices M1 M2 M3
solution sol0

	L	W	type	vgs	vds	vsb	ids	gm	rout	iratio
M1	100n	20u	nch	835.095m	13.246m	0	81.73u 110.149u	314.857u 310.251u	167.034 118.976	741.995m
M2	100n	40u	nch	536.754m	415.74m	13.246m	40.865u 158.091u	651.251u 2.023m	509.205K 8.994K	258.49m
M3	100n	20u	pch	-671.014m	-671.014m	0	-40.865u -194.494u	538.241u 1.561m	1.709M 11.193K	210.109m

Fig. 5. Example of Tk-based custom user GUI.

```
# Title o window
wm title .b "Biasing"
#Create buttons
frame .b.buttons
grid [button .b.buttons.reload -text "Reload window."
       -command "source table.tcl"] -column 0 -row 0
grid [button .b.buttons.compute -text "Compute" -command {
       compute $devices $biases $params $solution}] -column 1
       -row 0
# Define a procedure to compute devices parameters
proc compute {devices biases params solution} { ... }
# Create a table
frame .b.table; set _c 1; set _r 1
foreach par "L W type $biases $params iratio" {
  grid [label .b.table.par$par -text "$par" ] -column ${_c}
    -row 0; incr _c }
foreach dev $devices {
  # Labels of the table
  grid [label .b.table.dev$dev -text $dev ] -row ${_r}
     -column 0
  ...
  # Populate table values
    foreach par "$biases $params iratio" { ... } }
```

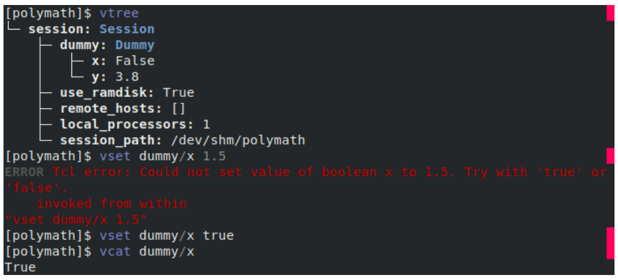
The custom GUI shows a window with a few text widgets for input data that specifies biasing conditions of a number of transistors, which are then displayed in a table. Some code is hidden for space limitations. The resulting window is shown in Fig. 5.

D. Data model

Our framework provides a set of data classes for nodes and parameters to be used in a tree data structure. The registered instances of these data classes are displayed automatically and can be navigated in the "Data Browser" dock in Fig. 4. The "dummy" node shown in that figure is an instance of the following "Dummy" class, which inherits from the "Node" class, while "x" and "y" are data-fields of types Boolean and float, all provided by Polymath.

```
from polymath.datamodel import Node, BoolParameter,
       FloatParameter
class Dummy(Node):
  x = BoolParameter()
  y = FloatParameter()
  def __init__(self, name, parent = None):
    super().__init__(name, parent)
    self.x = False
    self.y = 3.8
  def resource(self):
    return ":/icons/verif.png"
```

These templates of data structures accelerate the development of the application data model, since they provide most of the basic functionality required to integrate with the GUI and the Tcl shell. The Data Browser dock in the GUI can read and edit these parameters easily, while the type of data is automatically validated (Polymath provides templates for



```
[polymath]$ vtree
└─ session: Session
    ├─ dummy: Dummy
    │   ├─ x: False
    │   └─ y: 3.8
    ├─ use_ramdisk: True
    ├─ remote_hosts: []
    ├─ local_processors: 1
    └─ session_path: /dev/shm/polymath
[polymath]$ vset dummy/x 1.5
ERROR Tcl error: Could not set value of boolean x to 1.5. Try with 'true' or
'false'.
    invoked from within
"vset dummy/x 1.5"
[polymath]$ vset dummy/x true
[polymath]$ vcat dummy/x
True
```

Fig. 6. Example of “virtual file system” commands to access the Polymath data structure.

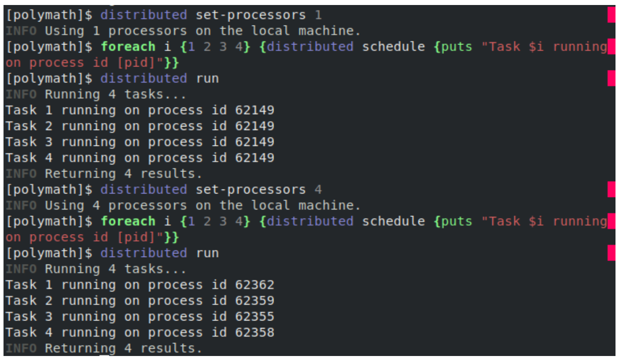
string, Boolean, integer, float, lists and dictionaries, but custom data types can be quickly created). Also, the Tcl shell provides some commands that access this tree data structure as a “virtual file-system”, with commands to list, set, print, etc, as exemplified in Fig. 6. Finally, the platform provides the functionality for any node in the tree to be exported to an XML file, which allows previously saved nodes to be loaded into the session without the need to code a specific serializer. Also, since all the nodes are children or sub-children of the parent “session” node (see the result of “vtree” in Fig. 6), the whole session is saved and reloaded easily with all the data that it stores. The adoption of the described data model does not impossibilitate an external data model, e.g. OpenAccess, to be adopted concurrently or to be wrapped around it. In most cases, it may be a better design choice to isolate large databases from the mentioned data model because the latter is inherently linked to the GUI, which may consume significant resources if the number of nodes is too large.

E. Event loops and distributed queue

One issue of integrating the Tcl shell, Tcl/Tk interpreter, Qt GUI with the core functionality executed when running the registered commands, is that many event loops (EL) need to operate together. For this reason, we propose the usage of an EL orchestrator. The EL orchestrator has its own EL that schedules and synchronizes the other ELs in a non-blocking fashion, i.e. the GUI is responsive when a command called from the shell is running. Simultaneously, the EL orchestrator also provides a distributed queue, which allows Tcl commands to run in multiple processors or threads within the same machine or in different machines through secure shell (SSH) connections. An example of usage of the distributed queue is shown in Fig. 7, where the queue is populated twice with 4 tasks that simply show in which process it is running, and runs with 1 and 4 local processors, respectively. This provides a very flexible approach for distributing workload on different processors or machines, while still using the commands created within the platform.

III. Continuous Integration/Deployment

So far in this paper we covered the aspects of developing EDA applications within the Polymath platform. However, an important feature of the framework is its capability of performing continuous integration and deployment (CI/CD). When creating new commands for the EDA applications, it is expected that the developer also creates automated tests which



```
[polymath]$ distributed set-processors 1
INFO Using 1 processors on the local machine.
[polymath]$ foreach i {1 2 3 4} {distributed schedule {puts "Task $i running
on process id [pid]"}}
[polymath]$ distributed run
INFO Running 4 tasks...
Task 1 running on process id 62149
Task 2 running on process id 62149
Task 3 running on process id 62149
Task 4 running on process id 62149
INFO Returning 4 results.
[polymath]$ distributed set-processors 4
INFO Using 4 processors on the local machine.
[polymath]$ foreach i {1 2 3 4} {distributed schedule {puts "Task $i running
on process id [pid]"}}
[polymath]$ distributed run
INFO Running 4 tasks...
Task 1 running on process id 62362
Task 2 running on process id 62359
Task 3 running on process id 62355
Task 4 running on process id 62358
INFO Returning 4 results.
```

Fig. 7. Example of distributed queue usage.

verify the correct behavior of such commands. Therefore, on each commit/push to the development branch of the version control system, e.g. Git, the automated tests run and the developer is notified if any of them fails. If all the tests pass, then the developer has the option to push the changes to the master branch. The tests are again run for the master branch and, if all of them pass once again, then the CI/CD system starts building the executables for distribution. To allow distribution to multiple platforms, the build cycle takes place inside Docker containers and finally the package can be released into a repository, allowing the users to update to the latest version. For all of our tests, we deploy to different versions of the same operating system which uses RPM as its package manager. The flow is exemplified in Fig. 1.

IV. Discussion and Conclusions

In this paper we presented Polymath, a RAD platform for EDA tools. The framework solves many problems which are recurrent when developing an EDA tool for wide adoption. Polymath is able to reduce the efforts in development by minimizing the amount of code required to link the core functionality of the tool to the user interface (command line and GUI). Also, a standardized data model is proposed in which the data is readily available to the user on the shell and GUI. The many aspects of the tool work concurrently through an event loop orchestrator that simultaneously provide a distributed queue for running tasks in multiple processors or machines through SSH. Finally, the integration and deployment of tools built on top Polymath is simplified by the usage of a CI/CD flow that automatically updates packages on a repository, speeding the process of fixing bugs and releasing new features. The platform has been successfully employed on the creation of a number of different modules for different EDA tasks, including design, characterization and verification of mixed-signal circuits.

Acknowledgments

The author would like to thank Carolina Metzler for the fruitful discussions. This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under projects UIDB/50021/2020 PTDC/EEI-EEE/31416/2017, and SFRH/BPD/116007/2016; and EU project Ecsel-783132-Position-II-IA.

References

[1] *Electronic Design Automation Tools (EDA) Market - Growth, Trends, and Forecast (2019 - 2024)*, Accessed: 2019-10-28. [Online]. Available: https://www.researchandmarkets.com/reports/4534513/electronic-design-automation-tools-eda-market.

[2] *Thomas Lee presents at The Internet of Everything: a Stanford Engineering symposium*, Accessed: 2019-10-28. [Online]. Available: https://youtu.be/1Cq9l7THciw.

[3] *History of Tcl*, Accessed: 2019-10-31. [Online]. Available: https://web.stanford.edu/~ouster/cgi-bin/tclHistory.php.

[4] G. Wood and H.-F. S. Law, "SKILL - an interactive procedural design environment," in *Custom Integrated Circuits Conference*, May 1986, pp. 544–547.

[5] *Si2 OpenAcess*, Accessed: 2019-10-31. [Online]. Available: http://projects.si2.org/?page=69.

[6] D. M. Beazley, "SWIG: An easy to use tool for integrating scripting languages with C and C++," in *Conference on USENIX Tcl/Tk Workshop*, Jul. 1996, pp. 15–15.

[7] *Qt website*, Accessed: 2019-11-1. [Online]. Available: http://www.qt.io.

[8] B. Welch, K. Jones, and J. Hobbs, *Practical Programming in Tcl/Tk*. Prentice Hall PTR, 2003.