2021 ASEE ANNUAL CONFERENCE
Virtual Meeting | July 26–29, 2021 | Pacific Daylight Time
Paper ID #32632

Program Encryption Toolkit: A Tool for Digital Logic Education and Undergraduate Research

Dr. Jeffrey Todd McDonald, University of South Alabama

Dr. Jeffrey "Todd" McDonald is a Professor of Computer Science in the School of Computing at the University of South Alabama. He received his Ph.D. in Computer Science from Florida State University in 2006, his Master of Science degree in Computer Engineering from the Air Force Institute of Technology in 2000, and his Bachelor of Science degree in Computer Science from the U. S. Air Force Academy in 1990. His research interests include program protection and exploitation, secure software engineering, and information assurance. He is a senior member of the ACM and IEEE.

Ms. Dawn McKinney, University of South Alabama

Dawn McKinney, a Senior Instructor and Curriculum Coordinator for Computer Science at the University of South Alabama, has been conducting research on Teaching and Learning for over 23 years and has co-authored over 25 papers which have been presented at SISCSE, ASEE, FIE, XP/Agile Universe, International Conference on The First-Year Experience, Southeastern Learning Community Consortium, Council on Undergraduate Research National Conference, and the South Alabama Conference on Teaching and Learning. As a leader in the university's Team-Based Learning effort, McKinney has been awarded funds for support, including travel, for the past seven years. She taught courses in China in 2013 and was awarded the highest award for teaching at the University of South Alabama in 2014. During the last three years, McKinney has participated in the Scholarship on Teaching and Learning program supported by the University of South Alabama and has been awarded funds to use for travel. During this time McKinney has collaborated with computer science faculty at several institutions and has co-authored papers submitted to both SIGCSE and ASEE.

Dr. Todd R. Andel, University of South Alabama

Dr. Todd R. Andel is a Professor and the Chair and of the Computer Science Department at the University of South Alabama's School of Computing. He received his PhD in Computer Science from the Florida State University (2007), a M.S. in Computer Engineering from the Air Force Institute of Technology (2002), and a B.S. in Computer Engineering from the University of Central Florida (1998). He was a prior faculty member of the Department of Electrical and Computer Engineering at the Air Force Institute of Technology from 2007 to 2012. He is a retired Major in the U.S. Air Force, serving over 23 years specializing in cyber systems defense, research, and education. He has published over 65 peer-reviewed papers and journals related to computer and information security, side-channel analysis, embedded systems security, network security protocols, and formal methods. Dr. Andel is a senior member of the IEEE and senior member of the ACM.

Program Encryption Toolkit - A Tool for Digital Logic Education and Undergraduate Research

Abstract: In this paper we present the Program Encryption Toolkit (PET)—a freely available Java-based graphical user interface that supports teaching and instruction for digital logic and advanced computer engineering concepts. PET has provided a vehicle for digital logic instruction and demonstrations targeting high school students that participate in a university partner school program, introductory Digital Logic courses in Computer Science curriculum, and for recruiting undergraduate researchers in cybersecurity related disciplines. We relate anecdotal success in using the tool for such engagements and context for student involvement as part of an object-oriented and agile software engineering project.

1. Introduction

Digital logic instruction is typically a core component of Computer Science (CS), Electrical Engineering (EE), and Computer Engineering (CpE) curricula. Most often, students are exposed to digital logic concepts early in their programs as a foundational steppingstone. CS students do not typically explore digital logic concepts further during their programs except as background knowledge for courses like Computer Architecture or Discrete Math. Instructors and students alike would benefit from a single tool that would provide a wide variety of functionality and support for visualization to reinforce concepts presented in such courses. In this research, we present the Program Encryption Toolkit (PET) and its associated graphical user interface (PETGUI) as a freely available resource for both educators and students who are learning early concepts related to digital logic. PET itself encompasses Java software contributed by undergraduate, master's and PhD researchers over a 15-year period and is used primarily to further research in circuit-based protection against malicious reverse engineering and subversion [1-11]. The code base has provided a means to also introduce software engineering concepts to students, including object-oriented (OO) analysis and design, OO design patterns, source code configuration, and team development.

In the remainder of paper, Section 2 provides background related to digital logic concepts and how PETGUI provides functionality for students to learn classroom concepts through specific use cases. Section 3 relates specific examples of using PETGUI in student learning contexts. Section 4 describes examples of how the software has been used to unify multidisciplinary research, engender undergraduate participation in research and development, and provide a recruiting tool for follow-on graduate studies. Section 5 provides conclusions and future work, with information on obtaining, installing, and using the PETGUI program.

2. Background

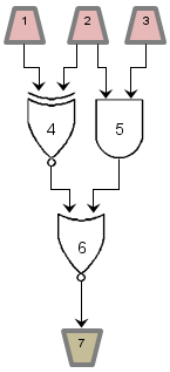
For digital logic courses, PETGUI offers a rich set of useful functionalities that can be integrated into course curriculum and educational outreach. The software is a front-end for a Java-based code repository that supports advanced experiments in program protection and exploitation related to hardware security that has been part of several master's and doctoral thesis topics [1-11]. The software integrates popular algorithms and synthesis tools such as UC Berkeley's

Espresso [12] and ABC [13] in an easy-to-use interface. Visualization from the graph-based Java yEd library [14] provides the ability to see relationships between circuit form, structure, and functional representations. It also supports advanced concept exploration with Binary Decision Diagrams (BDD) [15] and cryptographic properties of Boolean functions [16].

2.1 BENCH Format

PETGUI uses BENCH [17,18] format as the native representation for combinational and sequential circuit programs, which is a traditional hierarchical grammar for gate-level netlists. Figure 1 shows an example BENCH program and its schematic (as displayed by PETGUI). We chose BENCH format because of its ease in representing netlists and availability of the ISCAS benchmarks for studies in program protection. The basic format requires specification of the input/output (I/O) boundary of the circuit, interpreted in most-significant bit (MSB) ordering.

Schematic



BENCH

```
INPUT(1)
INPUT(2)
INPUT(3)

OUTPUT(6)

4=NXOR(1,2)
5=AND(2,3)
6=NOR(4,5)
```

Semantic TT:

```
123|7
-----
000|0
001|0
010|1
011|0
100|1
101|1
110|0
111|0
```

Full TT:

```
123|456|7
---------
000|100|0
001|100|0
010|001|1
011|010|0
100|001|1
101|001|1
110|100|0
111|110|0
```

Unreduced DNF/SOP:

```
F0(x1,x2,x3) = x1'x2x3' +
x1x2'x3' + x1x2'x3
```

Unreduced CNF/POS:

```
F0(x1,x2,x3) = (x1 + x2 + x3) *
(x1 + x2 + x3') * (x1 + x2' +
x3') * (x1' + x2' + x3) * (x1' +
x2' + x3')
```

Reduced ANF/ReedMuller:

```
F0(x1,x2,x3) = x2 ^ x2x3 ^ x1 ^
x1x2x3
```

Syntax:

```
F0(i1,i2,i3)=((i1^i2)'+(i2*i3))'
```

Figure 1: Circuit and Functional Representation in PETGUI

2.2 Basic Functions

Digital logic instruction often follows a typical sequence where topics are introduced as successive building blocks: 1) basics of electronic and digital systems such as voltage, current, resistance; 2) transistors (CMOS, nMOS, pMOS), logic gates, and truth tables; 3) Boolean algebra and equations; 4) circuit simulation and timing; 5) circuit schematics; 6) laws of Boolean reduction; and 7) canonical forms including minterms, maxterms, sum-of-products, and product-of-sums. Two-level simplification is typically explored later as Karnaugh maps and don't care conditions are introduced with the idea of end-to-end design based on smaller components. Typical components such as adders, subtractors, multiplexors/demuxes, encoders/decoders, and shifters are covered. Most introductory courses cover basic sequential circuit concepts including state machines, latches, flip-flops, and registers. High-level circuit description languages such as VHDL and Verilog are also typically introduced during first courses. PETGUI provides capabilities for illustrating each of these concepts within a unifying framework that supports standard course curriculum and textbooks such as zyBooks [19]. A full user's guide is provided online.[1] The predominant mode of PETGUI is for explaining combinational logic, but there is also rudimentary support for understanding sequential circuits as well. Flip-flops are treated as

[1] http://www.cis.usouthal.edu/~mcdonald/PET/PETGUI-UserGuide.v2.5.pdf

normal gate types (versus as a collection of gates themselves) and are interpreted accordingly in terms of producing input, output, and state behavior of the circuit.

2.2.1 Using and Creating Circuit Programs

PETGUI provides four ways to create or represent circuits: 1) from schematic, 2) from truth table, 3) writing BENCH text, and 4) loading pre-existing simple components or BENCH files. Pre-existing BENCH files can be loaded and modified/saved once in the PETGUI environment. Circuit programs can also be created using a schematic view. The general intuition for the builder follows many popular circuit-building tools and apps found online: Logic Friday [20] is one such Windows-based tool that has been popular for illustrating circuit reduction using Espresso. As Figure 2 illustrates, PETGUI provides a palette of basic gates that are chosen by type and placed onto a canvas. Input and output ports must be explicitly specified separate from intermediate gates. Simple drag and drop operations connect ports and gates together via wires. The builder provides different ways to layout the design visually, with hierarchical layout and orthogonal lines providing the traditional schematic view once all the gates and wires are placed. Once a design is finished, a user would validate the circuit to verify it follows a legal circuit format: all inputs are used once, the output port has only one source, all intermediate gate outputs are used at least once, and binary gates have at least two inputs/unary gates have one and only one input. The circuit can then be saved as a BENCH circuit format and automatically loaded as a new BENCH text panel within PETGUI.

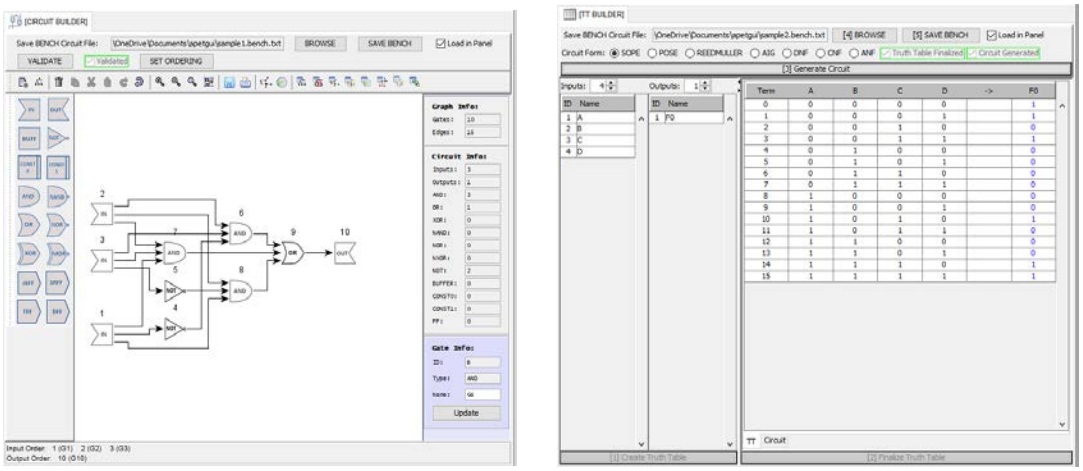


Figure 2 Schematic and Truth Table Builder in PETGUI

Figure 2 also illustrates the truth table builder in PETGUI which follows a five-step process. First, the number of inputs and outputs is determined. Once this is finalized a standard truth table is created with all 0 outputs for every output. Next, the truth table outputs are set to the on-set (1 outputs) and the truth table is finalized to define the desired function. Once the truth table is specified, a circuit (BENCH) structure is generated. Seven different options can be chosen to synthesize the circuit including reduced and unminimized sum-of-products (SOP), product-of-sums (POS), and xor-of-products (RSE). And-inverter-graph (AIG) format is supported through third-party tool generation provided by ABC. Reduction is accomplished through the Quine McCluskey algorithm [21]. Next a file is chosen to save the BENCH text as the fourth step and the final step will actually save and optionally load the BENCH file into a text panel within

PETGUI. The xor-of-products form is based on Ring Sum Expansion (RSE) and integrates Reed Muller's expansion technique for reduction [22]. RSE/Reed Muller forms are consistent with circuit functions expressed in algebraic normal form (ANF) equations [23].

2.2.2 Functional Representation and Formats

PETGUI provides support for illustrating the canonical versions and functional representation of circuits once they are in a BENCH netlist form. Of these, three basic unreduced equational forms of the circuit function can be viewed. Disjunctive normal form (DNF) or SOP, conjunctive normal form (CNF) or POS, and algebraic normal form (ANF) or RSE are provided as potential equational views of the circuit function. Figure 1 shows an example of a 3 input/1 output circuit with the PETGUI analysis provided different aspects: 1) semantic truth table (which shows only input/output), 2) full truth table (which shows all intermediate gate values), and 3) canonical equational forms. As a fourth option, users can also view the equation of a circuit as its structure or *syntax* would derive, seen also in Figure 1. In terms of reduction algorithms such as Quine McCluskey/Espresso and traditional Karnaugh Map (KMAP) views, PETGUI has capability to show both (seen in Figure 3).

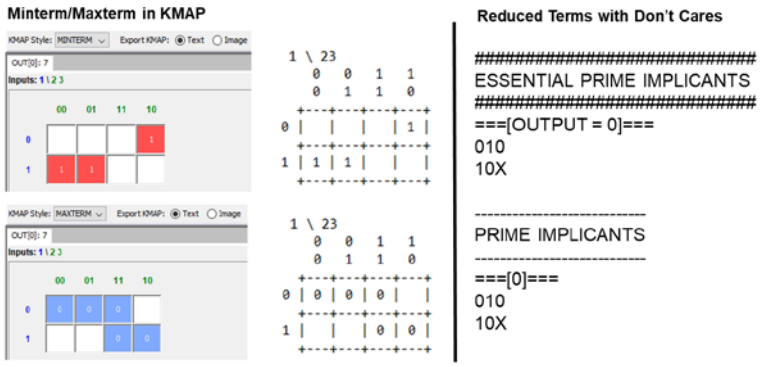


Figure 3: KMAP and Prime Implicant View in PETGUI

Prime implicants and essential prime implicants can be generated alongside a summary of minterms (1-producing input terms) and maxterms (0-producing input terms). PETGUI provides a view of the minterms and maxterms of the function in a traditional KMAP view, however reduction steps of the KMAP are not integrated directly into the interface currently (planned for future work). Figure 3 shows the KMAP of the circuit function in Figure 1 as an example and illustrates how the minterm information (010, 100, 101) could be reduced into smaller forms (prime and essential prime implicants) with don't care conditions (010, 10X). In practice, students could use the KMAP view as a basis to apply reductions and then check work using essential prime implicants. Through third-party tool integration such as ABC and Espresso, PETGUI can also produce several alternate representation forms of the circuit from its BENCH program. This includes a standard Programmable Logic Array (PLA) format, Berkeley Logic Interchange Format (BLIF), *structural* VHDL, and Verilog.

2.2.3 Transformation, Equivalence, and Simulation

PETGUI allows any BENCH circuit to be transformed into different (semantically equivalent) structures. This feature is useful for illustrating that functions can take on an infinite number of

structural forms (polymorphism) and that all canonical forms represent the same function. Figure 4 shows the circuit in Figure 1 transformed into SOP, POS, and RSE circuits in both unreduced and reduced (Quine McCluskey or Reed Muller) forms. PETGUI also has an equivalence checking function which can compare any two circuits to show whether their truth tables are equivalent. Circuits can be simulated to graphically show how input signals propagate visually, representing the same information found in the full truth table.

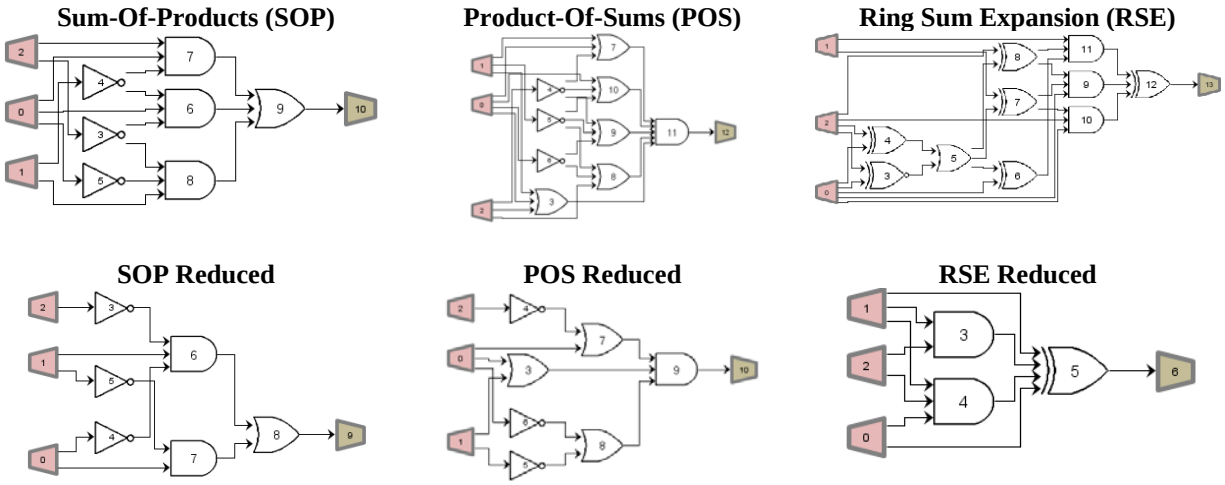


Figure 4: Canonical Transformations in PETGUI

2.3 Software Design

The PET software (and thus its graphical user interface front-end) are written in Java. Other historic research in circuit synthesis algorithms and their supporting tools have produced C or C++ programs that embody the most advanced implementations or realizations of the research. Until many of these functional features are available (or rewritten) in native Java format, PET uses a tool interface to access these third-party binary executables. PET takes advantage of four external tools: 1) Espresso [12] – a heuristic minimizer based on logic cubes; 2) ABC [13] – a tool specialized in synthesis and verification of synchronous hardware designs; 3) misII [24] –an interactive and batch-oriented tool for multi-level logic synthesis of combinational circuits; and 4) z3 [25] – a Satisfiability Modulo Theories (SMT) solver. PET interfaces with these software using a file/script generation process and a common known directory.

In terms of software construction, PET was designed using object-oriented (OO) analysis and design principles to maximize extensibility and enforce software engineering principles of encapsulation, modularity, reuse, abstraction, inheritance, and polymorphism. Figure 5 shows a UML class diagram of a very small subset of concepts from the domain layer of the application. The software itself has afforded itself as a teaching vehicle for classic software design patterns [26] such as Factory, Observer, Strategy, Builder, Flyweight, Façade, Decorator, and Composite. The OO nature of the software has also afforded many students the opportunity to contribute to the PET codebase for over a decade.

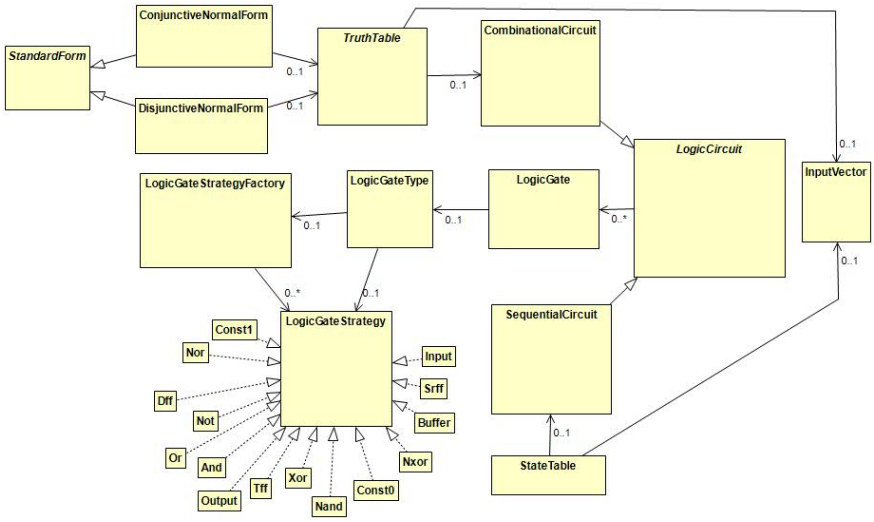


Figure 5: Key Domain Layer Software Classes

As Figure 5 illustrates, the fundamental domain concept in PET is an abstract Logic Circuit, which may be either Combinational or Sequential in concrete type. Logic Circuits have one or more Logic Gates (stored in a graph network) that have a type and other associated attributes. Combinational circuits have truth tables that relate their function, whereas sequential circuits utilize a state table that incorporates sequential component state signals into the input/output boundary of the circuit. The PET software is approximately 500K lines of source code at this point with over 20 contributors, almost all students. Almost every other part of the PET software depends on the core Logic Circuit abstraction, which speaks to the strength of the original OO analysis and design that has guided the software over the years. Refactoring has occurred several times since its inception, but the basic core constructs have allowed students to contribute various algorithmic implementations related to circuit manipulation and analysis in an incremental or agile fashion. Team development in PET has been controlled through standard source code configuration environments such as CVS and Subversion. Students work primarily in branches of the main source code, with a trunk that embodies the main version of the software.

2.4 Application Setup and Configuration

PETGUI is designed to be cross-platform for educational purposes in the sense that the basic functions used for introductory digital logic courses are written natively in Java. Research applications of the PET software require platform-specific versions of the third-party tools (discussed in section 2.3) that are used for certain minimization and synthesis operations. The default installation requires Windows for these special functions or to take advantage of the graphical interface to tools such as ABC. Windows, Linux, and Mac platforms that have a native Java Runtime Environment (JRE) with version 7 or above can run PETGUI as an executable Java Archive (JAR) file. Installation consists mainly of uncompressing a ZIP folder and running the application from there.

3. Educational Integration

In this section we relate anecdotal experience with utilizing PETGUI to enhance learning experiences related to digital logic, computer and electrical engineering, and core computer

science subjects. This has taken the form of K-12 STEM outreach, lectures in digital logic courses, and most recently in digital logic course integration.

3.1 K-12 STEM Outreach

At the University of South Alabama (USA), we have an active partner school program with the local community[2]. This has traditionally involved, in some part, bringing cohorts of 30-60 students at a time onto campus from local schools for hands-on lessons that are taught by staff, faculty, or graduate student assistants. Partner schools commit to whole-grade experiences, where all students in a given grade are introduced to specific topics related to the computing disciplines. These activities range from using Scratch The Cat for teaching introductory programming concepts to demonstrations of advanced software like Blender 3D used for creating animations. Associated summer camps have also focused on specific topic areas such as cybersecurity.

One of the first uses of PETGUI was in framing an hour-long activity to introduce digital logic concepts to high school students, mostly in grades 9-12. The general lesson plan for the session included 1) introducing what logic is in general, 2) introducing digital logic concepts, 3) explaining binary numbers and the concept of binary vs. decimal addition, and 4) introducing circuit programs in the form of gate netlists. Once the idea of normal logic statements is covered that introduce the concept of AND, OR, and NOT in the framing of valid arguments that are either true or false, students are then shown that computers and electronic components are based on similar concepts, but in the formal of digital logic components that use electrical signals for true/false values. After rudimentary gates and gate notation are introduced, the entire rest of the material is presented as hands-on activities using PETGUI. One of the local schools that houses a magnate program for college-prep in the engineering disciplines has shown specific interest in using the digital logic activity.

A major goal of the digital logic hands-on learning lab activity (HOLLA) is to show students that circuits are ultimately built, not from gates, but from collections of gates that form components. A part of the HOLLA discusses potential adversarial or malicious goals to hack hardware designs through illegal reverse engineering or subversion. To illustrate how circuit designs might be protected against such reverse engineering, the students are led through an exercise where they create their own (small) circuit component and use PETGUI functions to compose their custom component with other existing components to create a new circuit design. Figure 6 shows the design level view of the example and its corresponding gate level view.

Each student creates a 4 input/3 output component using the schematic circuit builder in PETGUI. An existing notional component (called *c17*) is then used to create the "front" part of the circuit. Each c17 component is 5 inputs and 2 outputs. Two of these are "merged" together to form a 10 input/4 output component, and then that component is "concatenated" with the student's custom 4 input/3 output component. The overall design thus creates a 10 input/3 output circuit. Students learn how traditional circuit designs are based on similar component building concepts and get the satisfaction of creating a unique part of the overall design. The

[2] https://www.southalabama.edu/colleges/soc/resources/partnerschoolsbooklet.pdf

very last part of the HOLLA illustrates how algorithms can be used to transform a circuit design into a version that takes more time or resources for reverse engineers to understand, with the goal of preventing disclosure of design information completely. The mathematical or cryptographic properties of the technique are not discussed as part of the HOLLA, but students walk away with the idea of how computer engineering, computer science, and cybersecurity can be tied together. It also provides an opportunity to promote cybersecurity as a field of study to high school students who are on target to enter college. The Digital Logic HOLLA was also one of the first learning modules that was setup for remote/off-campus instruction as part of the partner school program, mainly driven by COVID-19 restrictions.

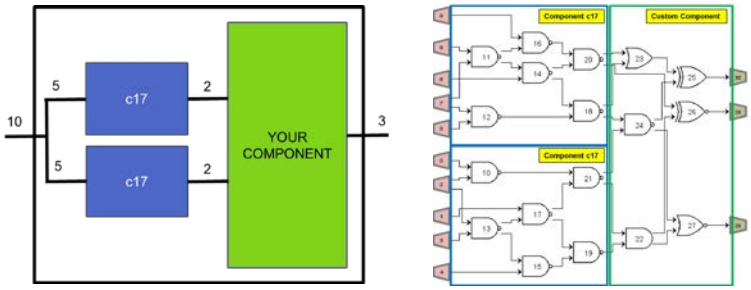


Figure 6: Component-Based Circuit Design for Digital Logic HOLLA

3.2 Guest Lectures

For normal undergraduate digital logic courses, PETGUI has been used for over 5 years to provide a one-time guest lecture for students taking our CSC-228 Digital Logic course. Most of the K-12 STEM HOLLA parts are used for this lecture, but with more focus on cybersecurity concepts and correlation with course activities. The lecture is always done toward the end of the semester, as students are working on a course project that involves design and implementation of a 3-bit CPU. Figure 7 shows the desired (end-product) of the course project which could be built up using constitute components. From semester to semester, the various "functions" and opcodes of the CPU change.

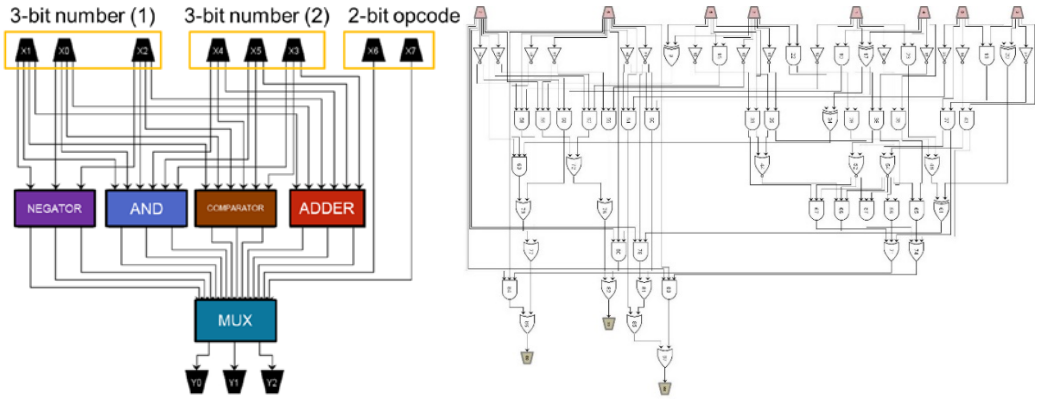


Figure 7: Example 3-bit CPU from PETGUI Component and Gate-Level Definition

The example in Figure 7 shows that two 3-bit numbers (A,B) are input along with a 2-bit opcode, which chooses 1 of 4 potential functions: NEGATE(A), AND(A,B), COMPARE(A,B), and ADD(A,B). The PETGUI guest lecture provides students with a context for 1) how larger

circuits are built up from smaller level components and 2) how there are situations where protection of the original design is important. This activity over the years has served several purposes: 1) it illustrated how cybersecurity might be relevant to students as they are studying digital logic concepts, 2) it gave a concrete example of what student research activities might involve 3) it unified course-long concepts of components and digital circuit design, and 4) provided a means to encourage undergraduate students to get involved with.

3.3 Course Integration

In a most recent offering CSC-228, PETGUI was integrated in the early part of the semester as part of the course curriculum. Students were introduced first to basic concepts of digital logic, such as the use of truth tables, simplification using Boolean Algebra, and the three basic gates (AND, OR, NOT). Students were provided a user guide, an introductory video on how to perform basic operations in the tool, and instructions on how to download, setup, and run the software. PETGUI was introduced in early lectures through demonstrations and small examples of creating circuit programs, displaying their schematics, and analyzing their function. Material for using PETGUI was integrated alongside zyBooks Digital Logic content and a Canvas course website. As anecdotal experience from this attempt to utilize PETGUI as part of the actual instruction, we incorporated an anonymous survey feedback to help gauge the usefulness of introducing a tool like PETGUI into the instruction.

Students (N = 39) completed the anonymous online survey after completing each of two assignments which required the use of truth tables, Boolean expressions, simplification, and the construction of circuit diagrams. The survey (seen in Figure 8) included 5 questions where students reported (1) their perceived level of difficulty completing the assignment, (2) the amount time they spent on it, (3) how much they enjoyed it, (4) to what degree they had the resources needed to complete the assignment, and (5) was an open-ended question allowing students to list the resources they used to complete the assignment. The two assignments were comparable, but different. The first assignment was completed before the introduction to PETGUI. The second assignment required the use of PETGUI. Figure 8 indicates that the students spent about the same amount of time on each assignment, enjoyed the PETGUI assignment slightly more, but the biggest differences were in level of difficulty and having the resources needed to complete the assignments. In the PETGUI experience, 23% of the students indicated the assignment was easier and 25% indicated that they had most or all of the resources needed to complete the assignment. For the pre-PETGUI assignment, students reported using a multitude of tools in order to complete the assignment such as spreadsheets, LibreOffice Writer, Word, pencil and paper, online resources (www.circuitverse.org), phone camera, scanner, Paint, and Notepad. After PETGUI was introduced, other than the PETGUI tool, students reported using far fewer tools, such as Paint (for labeling circuits), snipping tool (for capturing images from PETGUI), and Word (for putting it all together).

The anecdotal observations from this first attempt at course integration of PETGUI has provided many ideas for new features and adjustments to current ones. Overall, the attempt at integration has seemed positive and will more than likely influence how the course material is presented in the future. Given the initial positive instructor and student feedback, it provides a basis for

permanently integrating the tool alongside traditional course content. It also expands the opportunities for students to participate in future research using the PET software itself because they have a greater amount of time to use and understand the capabilities of the underlying theory and technology present in the user-interface itself.

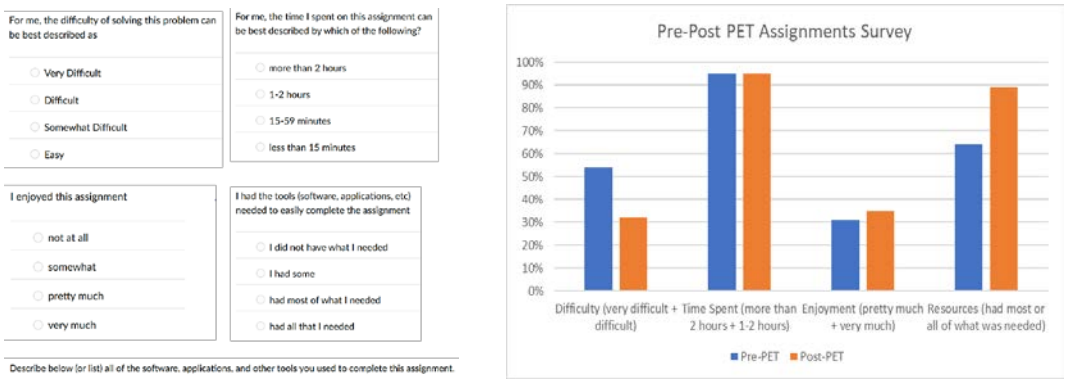


Figure 8: Pre/Post Anonymous Survey Questions and Results

4. Multidisciplinary Research Opportunities

Presentations of PETGUI in undergraduate settings has led to many opportunities to illustrate the multidisciplinary nature of research and potential ways students can learn hands-on-skills related to a real research project. In some cases, the presentation of cybersecurity concepts itself has caused many UG students to not only get involved as UG research assistants, but to pursue graduate level studies in the area. There are three ways in which use of the software has involved UG students:

1) Software engineering skills: students who see the demonstration of the software are also told that much work still needs to be done. Part of this work might involve volunteer or unpaid work as a research assistant but might result in paid/funded opportunities whether during the semester or as part of summer research programs. The KMAP functionality in PETGUI, for example, was added largely by student work after seeing the one-time lecture in the CSC-228 course. Three other UG students have volunteered to add features to PETGUI related to a component-based visual circuit builder (vs. gate-level), adding canonical form equations within the GUI (a current feature), and the newest feature, which is computing and showing cryptographic properties of small-input Boolean functions.
2) Program protection research: students that are intrigued by the computer/electrical engineering aspects of computer science research are drawn towards actual research in program protection, particularly with circuit programs in view. One such student, from a traditionally underrepresented group in Computer Science, was awarded a university-level summer internship to study circuit variation algorithms and made the subject their undergraduate honors thesis topic. A joint published paper [4] also resulted from the work as well as active participation by the student in an NSF-funded Secure and Trustworthy Computing (SaTC) grant that studied executable steganography for software protection.
3) Cybersecurity focus: several students who were intrigued by the cybersecurity aspect of the presentations went on to be involved in USA's DayZero cyber competition student organization, which focuses on training future cybersecurity professionals through

participation in cyber defense and capture the flag competitions. At least four students who were recruited as part of initial PETGUI presentations went on to be awarded scholarships under the NSF Scholarship for Service [28] program which funded graduate/master's programs with cybersecurity thesis concentrations.

5. Conclusions and Future Work

In this paper, we present the Program Encryption Toolkit Graphical User Interface (PETGUI) with a two-field purpose. One, to raise awareness of a freely available and open-source tool for use by those who are involved in curriculum development for digital logic and similar courses in Computer Science, Computer Engineering, and Electrical Engineering programs. Second, we relate our experience with how the tool has produced a marriage of educational outreach along with recruiting opportunities to encourage undergraduate students to get involved with research early in their academic programs. Future work for the application will focus primarily on demonstrating equational-level reduction steps that are involved with reducing a Boolean equation into simpler forms. The PET software currently has capability to apply standard Boolean logic laws to an equational form of a BENCH program [4]: additional work will be needed to create a graphical way to present such reductions through the GUI. The general requirements for this will center around showing how various laws affect the equational form of the circuit and reduce it into smaller versions. For installation and use, the PETGUI ZIP and user guide can be found online[3]. An instructional video for how to use PETGUI is also provided online[4]. The PET software is intended to be open-source and researchers interested in collaborating and extending the functionality of the software to support research studies in circuit program protection can contact Dr. Todd McDonald, jtmcdonald@southalabama.edu, for more information.

References

[1] Y. C. Kim and J. T. McDonald, "Considering Software Protection for Embedded Systems.," *Crosstalk: The Journal of Defense Software Engineering*, vol. 22, no. 6, Sept/Oct 2009, pp. 4-8.

[2] J. T. McDonald, Y. Kim, and A. Yasinsac, "Software Issues in Digital Forensics.," *ACM SIGOPS OS Review, Special Issue on Forensics*, vol. 42, no. 3, April 2008, pp. 29-40.

[3] A. Yasinsac and J. T. McDonald, "Tamper Resistant Software Through Intent Protection," *International Journal of Network Security*, vol. 7, no. 3, Nov 2008, pp. 370-382

[4] J. T. McDonald, T. L. Stroud, and T. R. Andel, "Polymorphic Circuit Generation Using Random Boolean Logic Expansion," *Proceedings of the 35th ACM/SIGAPP Symposium On Applied Computing (SAC'20)*, Brno, Czech Republic, March 30-April 3, 2020. doi: 10.1145/3341105.3374031

[5] J. T. McDonald, Y. C. Kim, T. R. Andel, J. McVicar, and M. Forbes, "Functional polymorphism for intellectual property protection," *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST 2016)*, May 5-7, 2016, McLean, VA, USA. doi: 10.1109/HST.2016.7495557

[6] T. R. Andel, L. N. Whitehurst, and J. T. McDonald, "Software Security and Randomization through Program Partitioning and Circuit Variation," *Proceedings of 1st Workshop on Moving Target Defense (MTD 2014)*, Scottsdale, Arizona, November 3, 2014. ACM Publishing. doi: 10.1145/2663474.2663484

[3] http://soc.southalabama.edu/~mcdonald/research.html

[4] https://southalabama.zoom.us/rec/share/6vl3spi-4qOJBZeQHyFMwKpCX8Ji-gAN6hvibxsEhNm_ZCHScBjFwwuuNAl3ozg.R8bNb6Y41mD7FPvS?startTime=1612289616000

[7] J. T. McDonald, Y. C. Kim, D. Koranek, J. D. Parham, “Evaluating Component Hiding Techniques in Circuit Topologies,” *International Conference on Communications, Communication and Information Systems Security Symposium (ICC-CISS-2012)*, June 10-15, 2012, Ottawa, Canada

[8] J. T. McDonald and Y. C. Kim, “Examining Tradeoffs for Hardware-Based Intellectual Property Protection,” *Proceedings of the 7th International Conference on Information Warfare (ICIW-2012)*, March 22-23, 2012, University of Washington, Seattle, USA.

[9] J. T. McDonald, Y. C. Kim, D. Koranek, “Deterministic Circuit Variation for Anti-Tamper Applications,” *Proceedings of the Cyber Security and Information Intelligence Research Workshop (CSIIRW ’11)*, October 12-14, 2011, Oak Ridge, TN, USA

[10] J. T. McDonald, E. D. Trias, Y. C. Kim, and M. R. Grimaila, “Using Logic-Based Reduction for Adversarial Component Recovery,” *Proceedings of the 25th ACM Symposium on Applied Computing (SAC’10)*, March 2010, Sierre, Switzerland.

[11] J. T. McDonald, Y. C. Kim, and M. R. Grimaila, “Protecting Reprogrammable Hardware with Polymorphic Circuit Variation,” *Proceedings of the 2nd Cyberspace Research Workshop*, June 2009, Shreveport, Louisiana, USA.

[12] R. L. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis," Memorandum No. UCB/ERL M86-65, 1986-06-05, Berkeley.

[13] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 70930. Available: http://www.eecs.berkeley. edu/~alanmi/abc/ [Accessed Feb. 15, 2021]

[14] “yFiles for Java Developer’s Guide,” yWorks. [Online]. Available: https://docs.yworks.com/yfiles/doc/developers-guide/index.html . [Accessed Feb. 15, 2021]

[15] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," in *IEEE Trans. Comput*, vol. 35, no. 8 (August 1986), 677–691. doi:10.1109/TC.1986.1676819

[16] T. W. Cusick and P. Stănică, Eds., *Cryptographic Boolean Functions and Applications*. Academic Press, 2009. doi: 10.1016/B978-0-12-374890-4.00001-X

[17] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," Proceedings, *IEEE Int. Symp. on Circuits and Systems (ISCAS)*; special session on ATPG and fault simulation, pp. 663-698, June 1985.

[18] M. C. Hansen, H. Yalcin and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," in *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72-80, July-Sept. 1999, doi: 10.1109/54.785838.

[19] R. Lysecky and F. Vahid, *Digital Design*, zybooks [online], 2014. Available at: https://www.zybooks.com/catalog/digital-design/ [Accessed March 1, 2021].

[20] P. Ravindran, "Logic Friday For Combinatorial Digital Logic Design," July 30, 2016. [Online]. Available: https://www.electronicsforu.com/buyers-guides/software-buyers-guide/logic-friday-digital-logic-design. [Accessed Jan. 18, 2021].

[21] E. J. McCluskey, "Minimization of Boolean functions," in *The Bell System Technical Journal*, vol. 35, no. 6, pp. 1417-1444, Nov. 1956, doi: 10.1002/j.1538-7305.1956.tb03835.x.

[22] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, 1st Edition. McGraw-Hill Science/Engineering/Math, 1994.

[23] I. Wegener, *The complexity of Boolean functions*. Wiley-Teubner, 1987. ISBN 3-519-02107-2.

[24] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Transactions on Computer-Aided Design*, CAD-vol. 6, no. 6, pp. 1062–1081, November 1987.

[25] L. De Moura and N. Bjørner, "Z3: an efficient SMT solver," In *Proc. TACAS'08/ETAPS*, pp. 337–340, March 2008.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st edition. Addison-Wesley Professional, 1994.

[27] A. Vahidi, "JDD: a pure Java BDD and Z-BDD library." [Online]. Available: https://bitbucket.org/vahidi/jdd [Accessed March 2, 2021].

[28] U.S. Office of Personnel Management, "CyberCorps: Scholarship for Service." [Online]. Available: https://www.sfs.opm.gov/ [Accessed March 2, 2021].