Top of Form

Bottom of Form

Top of Form

Bottom of Form

**LogicalCircuit**

יומן פעילות

1. שבוע המסתיים ב4/3. בחירת פרויקט – תצוגה גרפית של מעגל בוליאני בחירת תשתית לביצוע הפרויקט – נבחנו מספר אפשרויות: • שימוש בספריית IMGUI – ב C++ או ב Python. – קראתי חומר וכן התחלתי להכיר את הספריה. על פי המלצת המרצה אפשרות זו ירדה שכן אין היא אידיאלית בתור ספריה גרפית. • שימוש ב Schemadraw. היתרון של ספריה זו שהיא מספקת צורות מוכנות למעגלים לוגים וחשמליים. אולם זוהי ספריה סטטית אשר איננה תומכת באירועים ובנוסף חסירה עושר גרפי. • שימוש ב nodeeditor . למדתי Tutorial בנושא וכן צפיתי במספר וידאו אשר יוצר הספרייה סיפק. המסקנה היא כי זוהי חבילה מוגבלת אשר ספק אם מאפשרת את כל הנדרש, בנויה יותר עבור widgets מאשר עבור צורות גרפיות ומצריכה לימוד בהיקף נרחב מאוד. כיסוי הספריה וכן יישומים שלה מצריך צפייה בעשרות קטעי וידאו של כ30 דקות כל אחד וכן בהרבה מאוד דוגמאות קוד וגם אז ספק אם היא תתמוך בכל הנדרש. • שימוש ב PyQt5. גם כאן צפיתי בקטעי קוד והדרכות. אין ספק שספריה עשירה זו יכולה לתת מענה מלא אולם על פי המלצת המרצה בשל גודלה ועומקה של החבילה ויתרתי על אפשרות זו. • שימוש ב tKinter. דומה כי זוהי הפלטפורמה האידיאלית לביצוע המשימה. היא מספקת עושר גרפי כולל תמיכה באירועים דבר אשר יאפשר אינטראקטיביות עם המשתמש. בנוסף דומה כי ניתן להגיע לשליטה ברמה טובה מספיק על מנת לבצע לכל הפחות את החלק הבסיסי של הפרויקט בהיקף השקעה נמוך מאשר PyQt5. מסקנה – הפלטפורמה הנבחרת היא tKinter.

משך זמן עבודה – בחינת שלשת האפשרויות הראשונות ארכה כ 3.5 שעות. בחינת 2 האפשרויות האחרונות כ4.5 שעות, סה"כ 8 שעות עבודה.

1. שבוע שמסתיים ב 11/3 שינוי הבחירה ל חבילה nodeeditor על פי החלטת המרצה

מתברר מלימוד הקבצים כי עמ לשנות את הגרפיקה יש צורך בלימוד QtPy , צורת כל node מנוהלת בשני מקומות. מעטפת חיצונית בקובץ node\_graphics\_node בפונקציה paint. תוכן פנימי ב node\_editor\_widget בפונקציה initUI

בוצע שינוי הכותרות עמ לייצג שערים לוגים – הפונקציה addNodes בקובץ node\_editor\_widget.

הצבת ה nodes ע"ג החלון תתבצע באותו הקובץ בפונקציה addNodes .

טיפול ב socket מתבצע במחלקה Socket בקובץ node\_socket, node\_graphics\_socket

יש לשים לב לפונקצייה initSocket() שנמצאת במחלקה node\_graphics\_node

השליטה במיקום הכניסות והיציאות socket נמצאת בפונקציה node\_graphics\_socket.paint()

לימוד עבודה מול gitHub

סה"כ לחיוב – 6 שעות

1. שבוע המסתיים ב18/3

שליטה בכמות הכניסות והיציאות של כל שער נמצאת בפונקציה addNodes בקובץ node\_editor\_widget.

שליטה במיקום ה socket על גבול השער נמצאת בפונקציה getSocketPosition בקובץ node\_node .

איתחול ה sockets מתבצע ב קובץ node\_node בפונקציה initSocket.

בוצע שינוי של צורת האיתחול. במקור האיתחול הוא קשיח ומתבצע עבור כל הכניסות וכל היציאות בבת אחת, כלומר המשתמש יכול לשלוט במיקום של קבוצת הכניסות ולא של כל כניסה בפני עצמה וכן ביחס ליציאות. שיניתי את הקוד כך שניתן יהיה לשלוט על כל כניסה בפני עצמה

צפייה ב tutorials של דוגמת המחשבון.

סה"כ: 6 שעות.

1. שבוע המסתיים ב 25/3

עיבוד קבצים בפייתון, שילוב עיבוד הקובץ , הוצאת רשימת המעגלים מתוכו וציור השערים על המסך , עדיין לא בסדר הנכון ובלא חיבור ביניהם

שעות: 4 שעות

1. שבוע המסתיים ב 1.4 .

המשך עבודה על עיבוד קובץ ה Verilog. שליפת השערים למערך שערים, יציאות למערך יציאות, כניסות למערך כניסות

תיקון הבאגים וסיום עיבוד הקובץ, כמות שעות: 6 שעות

1. שבוע המסתיים ב8/4: הפיכת השערים לרשימה מקושרת על מנת למיין אותם בצורה קלה ונוחה בהתאם לכניסות והיציאות הלוגיות שלהם.

6 שעות עבודה.

1. שבוע המסתיים ב 15/4 – ללא התקדמות, נקיונות לפסח
2. שבוע המסתיים ב 21/4

סידור השערים על פני המסך בהתאם לסדר הלוגי שלהם (יציאות כנגד כניסות מתאימות) ולא לפי סדר הופעתם בקובץ ( Verilog מאפשרת רישום השערים בכל סדר שהוא).- היה אגוז קשה לפיצוח.

נסיון לשנות את התוכן הפנימי של כל צומת – בנתיים ללא הצלחה, הוא נקבע עבור כל הצמתים בבת אחת.

סה"כ 9 שעות עבודה.

1. שבוע המסתיים ב29/4:

לימוד חלק מה Tutorials וקריאה אינטנסיבית של הקוד ובסופו של דבר ביצוע:

* 1. הוספת רשימת הכניסות והיציאות לכל שער במקום כותרת פנימית אחידה לכל השערים
  2. קישור קשתות בין יציאה לכניסה המתאימה לה

אציין שהמיון והמשימות האלו היו הקשות ביותר עד כה

1. שבוע המסתיים ב 6/5:

טיפול בשמירה ובטעינה של קובץ דרך התפריט.

עד עכשיו התוכנית קראה קובץ hard coded והציגה אותו גרפית, עתה הוחלף המנגנון של המערכת הקורא דרך התפריט את קובץ ה Json בקריאה דרך התפריט של קובץ פרולוג (קובץ טקסט המכיל קוד בפרולוג) ושמירה של המעגל אל תוך קובץ טקסט (פרולוג).

בעיות שעדיין קיימות: החלפת ה default בתפריט מ Json לטקסט, ניקוי הלוח לפני טעינה חדשה

סה"כ שעות: 12 שעות.

1. שבוע המסתיים ב 13/5:

שינוי פילטר שמירה וטעינה מ json ל txt כך שיתאים לקבצי verilog לצורך שמירתם וטעינתם.

הוספת כניסות המעגל לתצוגה הגרפית בתור צמתים בנוסף לשערים הלוגים

הוספת תוכן הסוגריים בשורת ה module בשמירה לקובץ verilog בהתאם לכניסות וליציאות של המעגל כולו (כלומר כניסות ויציאות ולא שערים לוגים) כך שהמעגל ישמר כולל כניסות ויציאות או

עדיין לא טופל – מחיקה של כניסות לוגיות

עדיין לא טופל – רענון המסך במקרה של טעינת קובץ לאחר שקובץ כבר מוצג.

סה"כ 8 שעות.

1. שבוע המסתיים ב 20/5

ביום שישי לא הרגשתי טוב ולכן סיימתי את העבודה של שבוע זה רק בראשון 22/5

תיקון שני באגים מרכזיים:

* 1. במקרה של שער או קלט למעגל המקושר ליותר מכניסה אחת הוצג רק קישור אחד – תוקן
  2. טעינה של קובץ חדש (קובץ טקסט בנגוד לתשתית של קובץ Json שביטלתי אותה) הוסיפה שערים לוגיים ולא העלימה את הישנים. – תוקן

סה"כ 4 שעות.

1. שבוע המסתיים ב 3/6.

הוספת תוכן הסבר לפרויקט (למטה)

עבודה על באג – מחיקת צומת קלט וטעינת הקובץ לאחר שמירתו – התיקון לא הושלם במלואו עדיין.

7 שעות עבודה

**ReadMe file**

The Logical Circuit Visualizer (LCV) project is aimed to read and graphically present a Logical (Boolean) electric circuit programmed in the verilog programming language.

The LCV was written in python and is based over the **NodeEditor project (https://www.blenderfreak.com/tutorials/node-editor-tutorial-series/ ) written by Pavel Krupala** after customizing modifying and enhancing it to support the LCV needs.

The NodeEditor project is using Python PyQt5 package.

**Logical Circuit Visualizer functionality**

1. **Reading and parsing a valid verilog file**

The LCV will parse any given valid verilog code even if the order and relations in between the gates are not in their right logical order as the verilog programming language supports this option.

The LCV will identify the right position of each gate based on its input and output relations.

1. **Graphical Presentation Of The Verilog Code**

The LCV will present the Boolean circuits graphically**.** A Boolean (logical) gate will be presented as a node in the node editor.

The size of the circuit or number of boolean gates are unlimited.

The boolean gates alignment on the screen will be done according to their logical order based on their inputs/outputs and not over their chronological order as appears in the input verilog code file.

Circuit zooming capabilities are supported using the mouse wheel.

The LCV supports partially programmed circuits as long as they are in accordance with the verilog programming language.

1. **Limited Graphical Editing Capabilities**

The LCV provides a limited graphical editing capability as its main goal is to visualize a verilog circuit code. One may delete a gate (or several gates) in the circuit and save the new circuit back into a verilog text file.

The LCV will translate the new visual circuit back into a valid verilog code.

If the newly saved verilog file is now loaded back to the LCV, it will be displayed graphically with the changes made to it.

The user may also delete an edge connecting two gates but this deletion will not be saved. The reason is that the LCV ignores the user edge connecting and will perform the correct edge connecting between the gates based on their right entries and exists and will not allow a wrong connection.

The LCV does not support adding new gates or modifying existing ones.

**Assumptions**

1. The input file is a valid verilog file.
2. The circuit will contain no more than 2 entries as in a logical circuit. Circuits with more than a pair of entries are parts of larger circuits and not an atomic circuit.
3. The logical gates supported are the basic logical gates which exist in the verilog language. I.e. AND, OR, NOT, XOR, NAND, NOR. Private customized gates created by users are not supported however they may be easily added to the LCV (though will require some minimal coding).

**Known Limitations**

1. The Boolean Visualizer project is not a verilog compiler. It doesn’t create any native or machine code, it doesn’t provide compilation errors or debugging capabilities.

It assumes a valid verilog code text file as an input.

1. If the input code is not a valid verilog program, the results are undefined and depend on the syntax errors in the file. Gates which are properly defined will be presented properly while gates that were incorrectly defined (in verilog terms) might just be ignored or presented without the proper links inside the circuit (depends on the type of syntax error).

On some faulty cases and if the code is not at all a verilog code, the Boolean Visualizer will ignore the code and present nothing.