Abstract Due to increasing size and complexity of modern hardware designs, the challenge of identifying a piece of design becomes increasingly difficult. This is especially true, if no documentation is available. This factor has a direct impact on the time that is needed to get familiar with a design. In extreme cases, the design is rendered useless for the user. A hint on what hardware category the design belongs to, would accelerate the process of familiarization. This work considers, if it is possible to categorize hardware designs, that are given as Hardware Description Language, on basis of their structure. The elaborated algorithm is able to categorize a given design in X seconds, with an accuracy of S.

Kurzfassung Mit steigender Größe und Komplexität von modernen Hardware Designs, wird es zusehends herausfordernder die Funktion des desselben zu identifizieren. Vor allem trifft dies zu, wenn keine Dokumentationen zum Design verfügbar sind. Dieser Umstand wirkt sich unmittelbar in einer erhöhten Einarbeitungszeit aus. In Extremfällen muss der Anwender das Design wegen Unbrauchbarkeit verwerfen. Ein Hinweis darauf welcher Hardware Kategorie das Design angehört, würde den Einarbeitungsprozess beschleunigen. Diese Arbeit untersucht, ob es möglich ist Hardware Designs, die als Hardware Description Language vorliegen, anhand ihres strukturellen Aufbaus zu klassifiziern, und in Kategorien einzuteilen. Mithilfe des erarbeiteten Algorithmus ist es möglich ein Design innerhalb von X Sekunden, mit einer Sicherheit von Y zu klassifiziern.

1 Introduction

As Embedded Systems find their way in an increasingly wide field of applications, with growing demands to performance and relieability, the underlying Hardware Designs also gain in diversity, complexity and size. Since it is nearly impossible, even for simple Designs, to determine the function of such without proper documentation, a possibility to extract information directly from the Hardware Description Language representation of the Design, seems to be a welcome aid. Such a hardware design classification algorithm also allow services, which automatically handle aforementionend designs (for instance online sharing plattforms like glsoc) to assign categories without relying on user input.

This work tries to achieve a classification of hardware designs by analyzing the connection between logical cells. The count of specified two to one and one to one connections (examples depictioned in illustration), are translated into a standardized match vector. It is expected that the match vector of similar designs gather in clusters. These clusters can then be understood as hardware categories. Using a methodolodgy like this implies, that the location and meaning of those clusters first have to be identified, by analyzing a great amount of well known hardware designs. The clustering is also in scope of this work, and is achieved by a web crawler, which gathers already categorized designs from the open source sharing plattform glsoc. These designs are then synthesized by the open source synthesis tool glsyosys and handed over to the match algorithm, which eventually determines the match vector. The so gathered set of match vectors are then grouped into clusters.

The result of this work is a ¡command line application¿ which is able to categorize large designs (XXX Cells) in X seconds with a certainty of YThe following chapters elaborate on the development, set of problems and other possible applications of this work.

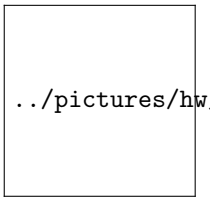

Figure 1: Register Transfer Level depiction of the proposed Hardware Trojan nets

2 State of the Art

Literature regarding this topic is very scarce, if not non-existent. At the time this work has been released, no other publications which attempted to establish an algorithm to automatically identify hardware designs, could be found. Papers which worked on topics remotely relatable with the topic of hardware identification, mostly presented methods to identify hardware trojans in a given design. This is mostly achieved via a functional analysis, where the design is simulated and tested for a certain behaviour. Since our method aims for a structural analysis, these publication are hardly compareable. One publication used a combination of structural- and functional analysis, to identify hardware trojan design patterns. This methodolodogy should be further looked upon.

2.1 Detection of Hardware Trojans

In [literature] X net types are defined, which are typical for hardware trojans. shows those proposed Hardware Trojans nets. Similar to our proposed method, the count of these nets are determined.

The publication states, that the count of certain net types is not sufficient to identify the presence of a hardware trojan. They introduce additional functional tests to further improve their detection rate.

Figure 2: .JSON File Format Description

3 Metholodgy

We use Scrapy, a Python web scraping library, to automatically obtain design files and corresponding meta-information for each design that is published via oc. We are interested in the following data:

1. url to the compressed design archive
2. The name of the design project
3. The hdl in which the design is specified and implemented
4. The category in which a design is listed on oc
5. The hdl files of the design

3.1 Learn Process

3.1.1 Scraping OpenCores project web sites

The scraped data is temporarily stored into Python class objects. The information in these objects are written to a .JSON file, in order to make them persistent (such that we do not have to scrape the entire oc web site every time we use our classification framework). For this .JSON file, a specific format has been defined, so information can be imported from other sources then scrapy aswell.

Figure fig:jsonFormat shows an example of the content of the .JSON file.

3.1.2 Downloading OpenCores projects

The Scrapy python module provides the functionallity to automatically download and store files that are associated with a Hardware design project, to a predefined folder. Since oc requires an account to be able to download hardware designs, we use a scrapy internal function to send a POST request to the oc webpage, in order to authorize ourselfs.

Because all project files from the oc database are provided as tar.gz archives, we introduced an additional decompression step.

3.1.3 Decompressing design files

Projects from oc are solely provided as *.tar.gz compressed archive. The python module tarfile enables the extraction of those archives. Additionally to the decompression, we sort the project files into folders corresponding to their associated hardware category (as stated by oc). After that, the file endings of the files are analysed. If those endings indicate a HDL file, the path to this file is added to the aforementioned .JSON file, in order to be able to address the design files of a project in later steps.

3.1.4 Reading designs into synthesis tool

After the design has been decompressed, it is time to let the synthesis tool yosys read the design files and synthesise them into a text file format. In order to do this, yosys expects a yosys script file, which holds all yosys commands that should be executed on a set of files. This script file can either be provided by the user before a programm run, or a generic one can be generated automatically during a programm run, according to the files that have been provided with the .JSON file (lst:example).

The synthesis tool's frontend is chosen based on the language of each design file. For vhdl and SystemVerilog, the verific frontend is used. For Verilog, we use the read$_verilog frontend. Once any combination of hdl files has been loaded, a synthesizer unattempts to generate a$

Since yosys does not support automatic dependency recognition of vhdl files, a custom solution had to be found, to determine the load order of vhdl files (in the case that the user decides that yosys scripts should be generated automatically). To accomplish this, we slightly modified the Vunit VUnit python project, which offers a function to return the vhdl files in an ordered list. The vhdl files can then be loaded in the order dictated by this list.

3.1.5 Naming designs

Each design that is read by the synthesis tool is named after the project from which the design files are downloaded. From then, the design name is the main reference for each design and serves as identification feature in all subsequent steps.

3.1.6 Matching designs against standard pattern vector

3.1.7 Calculating clusters of design match vectors

3.2 Verification

4 Discussion

5 Conclusion