DATE: 2018/1/9

Department of Electrical and Electronic Information Engineering		ID	M143276
Name	Yumi Matsuoka		

Supervisor	Shuichi Ichikawa Naoki Fujieda

Abstract

Title	Preliminary evaluation of hardware obfuscation with LLVM and high level synthesis tool

In recent years, software copyright infringement became a serious problem. Protecting software from this kind of threat is important. Obfuscation is a technology that converts protected objects such as programs and circuits so that they are functionally equivalent and have a complicated structure to prevent analysis. Another protection technology is the conversion to hardware. Hardware is considered to be difficult to analyze, steal, and tamper compared to software. In recent years, hardware generation by high-level synthesis (HLS) has attracted attention. By using HLS, a designer can design at a higher abstraction level than RTL, and it is possible to reduce the time and effort required for design.

This research adopts oLLVM, which is an obfuscation tool developed by Pascal et al. oLLVM works with LLVM, which is a compiler basis. Since oLLVM obfuscates the intermediate representation (IR), it is possible to obfuscate various languages by changing the front end processor. oLLVM outputs the obfuscated intermediate representation, which is then processed by a back end processor. The obfuscated IR is converted to obfuscated C program by using C back end which converts the IR into C language. The obfuscated C program is then synthesized by Vivado HLS which is a high-level synthesis tool for hardware evaluation.

CHStone was adopted for the evaluation, which was developed by Hara et al. (2009) for C-based high level synthesis. CHStone is composed of 12 programs selected from various application areas including arithmetic operation, media processing, and security. However, CHStone involves some problems, since it was not developed for Vivado HLS. The main problem is that Vivado HLS limits pointer support. This study deals with the methods to avoid the use of unsupported pointer.

The items of software evaluation include execution time and code size. In the evaluation results, the execution time of AES_F (the control flow flattened cryptographic algorithms AES) was 12.7 times larger than the original AES. In case of BLOWFISH and SHA, this ration was 3.6 and 6.7, respectively. The increase of the execution time depends on the application, while the ratio of code size is around three for each application. This result might be caused by the fact that AES heavily uses for statements and switch statements. From each CHStone code, the number of for statements used is in the order of AES, SHA, BLOWFISH. Therefore, it is considered that control flow flattening is working correctly.

The respective code sizes of AES, BLOWFISH, SHA became 5.5, 7.8 and 3.8 times larger by adding bogus control flow, while the respective execution time was 3.7, 3.8, and 4.4 times. From this result, it was found that BLOWFISH has the smallest increase in code size with respect to execution time.

The hardware evaluation results presents that when control flow flattening is applied, FF and LUT usage increased in all programs. Also, in some applications such as AES that applied bogus control flow or instruction substitution, the amount of hardware decreased.