



MeXT-SE User Guide

Version 1.0

Date 3/22/2020

Table of Content:

1. Introduction
2. Tool Flow
3. Dependencies & Installation
4. File hierarchy
5. Opening MeXT-SE
6. Development Instruction
7. Add XML file
8. Enable Security
9. Adding Policy File
10. Adding Rules
11. Generating TCL script
12. MeXT-SE Quick Start
13. Appendix

1. Introduction

The Multi-processor Exploration with Security Extension (MeXT-SE) is an SoC development tool designed in the Smart Systems Lab at UF. MeXT-SE can generate platform-independent secure multi-processor systems from a high-level abstraction. The user will provide abstract and concrete specifications of a system such as the number of processors, memory size, hardware IPs, operating system, target device, etc. The tool uses this knowledge to generate the appropriate hardware design script for the required vendor. The input to MeXT-SE is an XML file with abstract specification and the output of this tool is a platform-specific tcl script that can be directly run on Vendor tools (e.g. Xilinx Vivado, Intel Quartus, Accellera SystemC) to generate a bitstream. While generating the hardware design, MeXT-SE enforces a flask security framework to govern access control between components of different security domains. This feature is imposed transparently to the design to ensure security on the generated SoC.

[Note: The purpose of MeXT-SE tool is not to replace Vendor tools. Rather, MeXT-SE eases the development of secure SoC designs by enforcing access control mechanism on the hardware design by default]

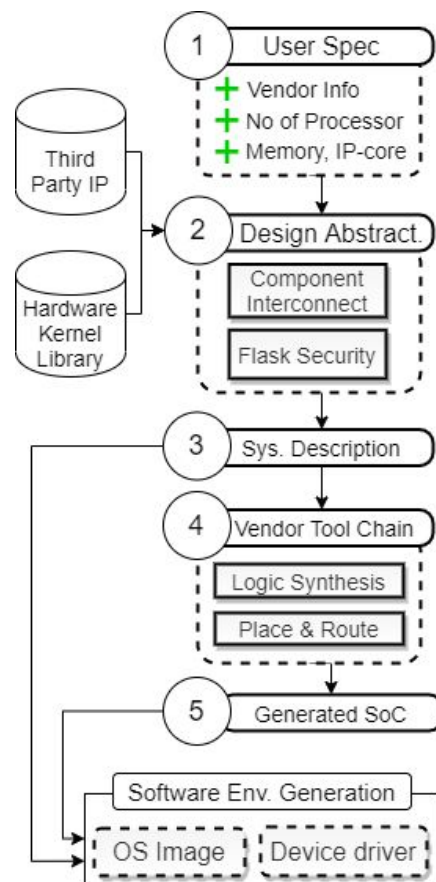


Figure 2.1: MeXT-SE design flow

2. Secured System-on-Chip

Figure 2.1 shows the target system-on-chip sought to be generated by mext and operate safely. The tool starts with a user-defined abstract and concrete representation (XML format) of a system and ends up generating a final hardware design with enforced countermeasures for preventing unauthorized access of hardware IPs in the SoC. While setting up the communication network, an access control mechanism that inherits MAC-based authentication policies, found in the flask security architecture, is enforced directly in the hardware design. The architecture of the generated architecture design is shown in figure 2.1.

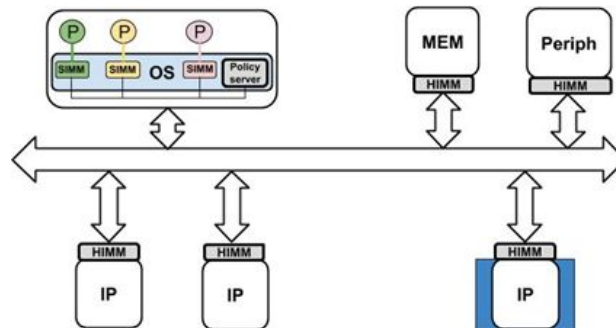


Figure 2.1 Architecture of the Secure SoC design generated by MeXT. Here the Hardware module manager is added transparently by the tool that enforces flask security on hardware.

The flask security framework is implemented as a decentralized hardware/software architecture with the Hardware IP Management Module (HIMM) governs access control at the IP level, while the Software IP Management Module (SIMM) manages policies inheritance of security contexts and queries the host kernel security server for associated permissions (See appendix for more details).

The HIMM consists of an internal enforcement function and an Access Vector Cache component (AVC) to cache the last queries and ensure that policies check are handled locally. Each HIMM maintains a map of security context labels for the corresponding hardware module it manages. It implements a custom circuit, the “Access Enforcement Function,” which guards access to the hardware modules according to the host kernel MAC policy. Upon receiving an access request to the hardware module, the SIMM module on the host CPU queries the host kernel security server for associated permissions. The server consults its MAC policy and returns associated permissions, which are then sent to the respective HIMM module for future access requests.

3. Dependencies & Installation

MeXT-SE is designed to run on all OSes. The dependencies of MeXT-SE are listed below:

- Python3
- tkinter
- Pillow (for MAC & Windows)

To install the tkinter for python3 on Ubuntu:

```
~$ sudo apt-get install python3-tk
```

MeXT-SE does not have any additional installation instruction. Once the dependencies are resolved, download the repository in your local directory. To download the repository go to the following link and download.

<https://github.com/smartsystemslab-uf/MeXT-SE>

Command prompt download command:

```
~$ git clone https://github.com/smartsystemslab-uf/MeXT-SE.git
```

Follow step 5 to launch the MeXT-SE gui.

4. File Hierarchy

Once the download is complete browse into your MeXT-SE directory. The section gives an overview of the file hierarchy of the repository. You will see a file hierarchy like below.

MeXT-SE

|
design folder* | *images* | *policy-folder* | *src

Design folder: holds example design files (*.xml). The generated tcl script is also stored in this directory.

Images: tool image icons.

Policy-folder: holds policy files (.te). The folder also holds the tool generated SELinux policy scripts.

Src: source code of the tool. Written in python.

5. Opening MeXT-SE

Execute the main file, found at <ToolLocalDirectoryPath>/MeXT-SE/src/main.py.
Here <ToolLocalDirectoryPath> is the path where you have downloaded the directory.

```
~$ cd /ToolDirectoryPath/MeXT-SE/src/  
~$ python3 main.py
```

This should open up the main window of the MeXT-SE tool. The next few sections provide information on the functionality of MeXT-SE. Section 12 comprises a getting started guide. Users are highly encouraged to start from there.

6. Development Instructions

As mentioned before, the MeXT-SE tool takes XML file as input. To develop a new application from scratch, please create an XML file with system instruction. Please look at example designs found in the <design-folder> to get familiar with the format. Once the XML file is created please follow step 7 through step 11 to generate the tcl script.

7. Adding XML file

On the MeXT-SE tool window, there are three subwindows. The bottom window shows the log messages of the current operation. The top-left window shows the example files. And the top-right window shows the hardware-design tree generated from the XML file. An XML file can be added to the project, which will outline the overall architecture of the processor. Click the icon shown in figure 7.1. In the pop-up window please browse to the correct directory and select the XML file. MeXT-SE will load the XML file and automatically parse the relevant information from the file. The generated hardware design tree can be seen in the window below it. The user can click on the Add module button(green plus sign) to add a new component on the XML file.

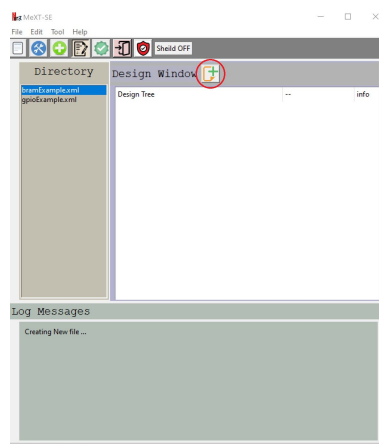


Figure 7.1: Adding an XML file

8. Enable Security

The flask security architecture is not turned on by default in MeXT-SE. When turned off, the tool will generate hardware design script without security enforced. To enforce flask security please click on the shield icon indicated in figure 8.1.

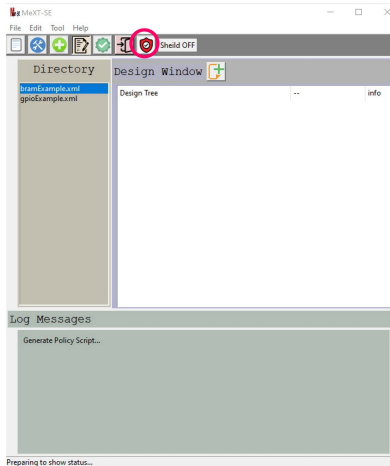


Figure 8.1: Shield Toggle

9. Adding Policy File

The MeXT-SE tool generates hardware design with enforced flask security architecture. The tool can be used to set up MAC-based security rules for the OS running on the embedded processor. The tool will generate the necessary bash scripts to install these rules on the OS. To add a policy file to the current project, please navigate to **Tools>Add new policy file** and browse to the intended policy file(*.te) or click the icon shown in figure 9.1 and browse to the file from there.

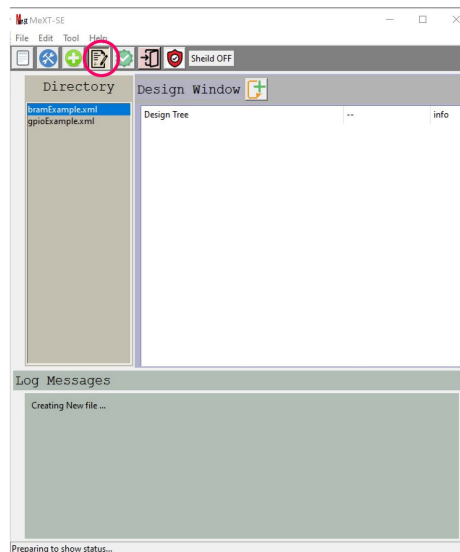


Figure 9.1: Adding a Policy File

10. Adding Rules

The user can add rules directly to the policy file from the tool. After adding a policy file, the user can add new rules by going through “**Tool>Add new rule**”. There are 3 permissions (read, write, execute) that can be assigned to a given subject for a given object. There are two types of class: file and process. The command types are: allow, neverallow, audit2allow.

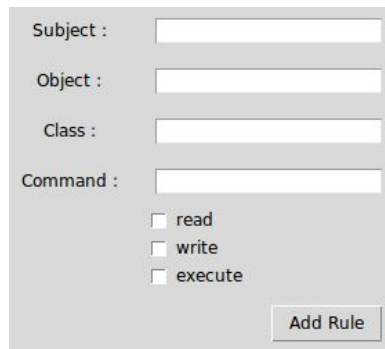
A dialog box titled "Add new rule" with a light gray background. It contains four text input fields labeled "Subject :", "Object :", "Class :", and "Command :". Below the "Command :" field are three checkboxes labeled "read", "write", and "execute". At the bottom right is a button labeled "Add Rule".

Figure 11.1: Add new rule window

Once the rules are added, please click “**Tool>Generate policy scripts**” to generate bash scripts to load policies to the OS (shortcut: green tick button on the toolbar).

11. Generating TCL Script

Once the XML and Policy files have been provided, and the rules have been defined, a TCL script can be generated to be used in Vivado. To generate the TCL script, please go to “**Tool>Run Synthesis**” or click on the icon in figure 11.1.

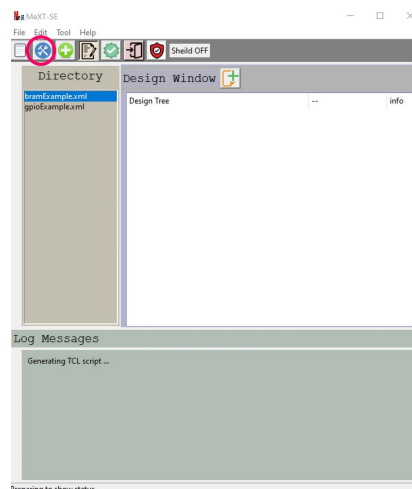


Figure 11.1: Generating TCL Script for Vivado use

This will generate the final tcl script that can be sourced from a Vivado terminal to generate the bitstream file.

12. MeXT-SE Quick Start with Xilinx Design Flow

This section provides a quick start on MeXT-SE tool with an example design.

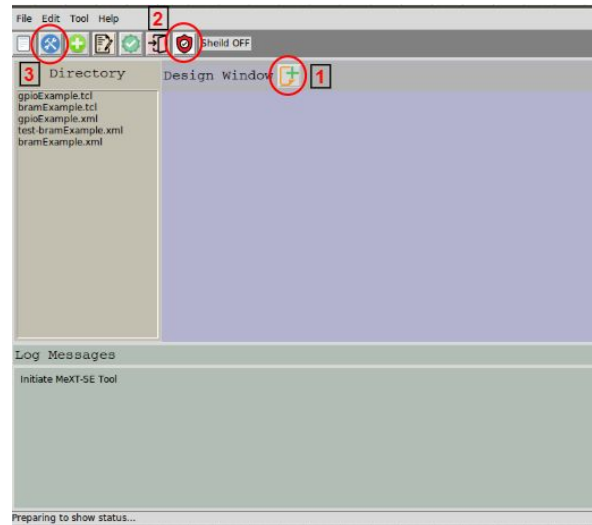


Figure 12.1: MeXT-SE tool GUI.

- (a) Download and save the MeXT-SE repository in your local directory. The IP repo for the custom security modules required to model flask security framework is included in the repository. The tool will require the absolute path of this IP repo to include it in the generated TCL script. To add this open `XilinxSecureScript.py`. Replace line number 35 with the absolute location of the ip-repo.
("set_property ip_repo_paths <your abs path> [current_project]\n")
- (b) Open up GUI.
`~$ cd /ToolDirectoryPath/MeXT-SE/src/`
`~$ python3 main.py`
- (c) Now we add the XML file comprising the hardware description. Click the icon marked as (1) shown in figure 12.1. In the pop-up window please browse to the <design-folder> and select `gpioExample.xml` and click open. MeXT-SE will load the XML file and automatically parse the relevant information from the file. The generated hardware design tree can be seen in the window below it.

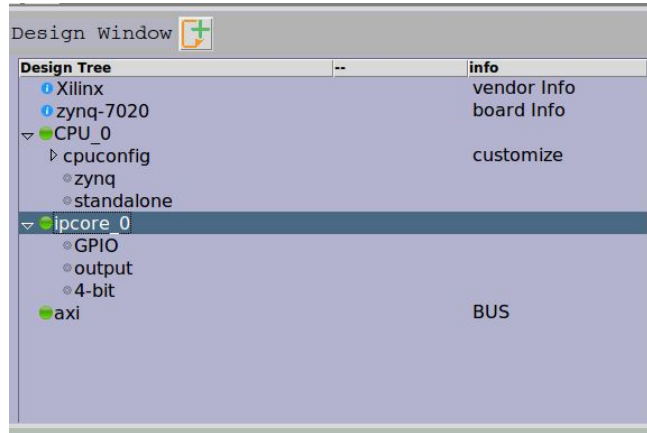


Figure 12.2 Generated design tree example

- (d) All the necessary components are already added to the example XML file. So we do not need to add any new component. Please enable flask security framework as described in step 9 [Figure 12.1 (1)].
- (e) To Generate the script please go to “**Tool>Run Synthesis**” or click on the icon (3) in figure 12.1. This will generate a gpioSecureExample.tcl on the <design-folder>. Now open vivado and on the console source this script using the following command.
- ```
<source gpioSecureExample.tcl>
```
- This will generate the hardware block design on vivado. From here please generate the wrapper vhd file and generate bitstream following the vivado design flow.
- (f) MeXT-SE allows user to define rules directly from the tool. That can be used later on the embedded OS to setup rules by an authorized user. Please follow step 10 and 11 for that.
- (g) To automate the generation of the OS using petalinux, please go to the “MeXT-SE/OS\_generation/Xilinx” directory. There is a README file which describes a complete guideline and requirement for OS generation steps. Also, there is a “petalinux.sh” script file which will automatically generate the OS images for Xilinx platform. But before running the script, please read the README file.

**Note:** This is to notify that the MeX-SE too is under continuous development. At present it provides support for only a specific set of applications. However, we will be adding more functionalities in future.

## 13. Appendix

### (a) Flask Architecture:

In MeXT-SE, we inherited flask architecture in hardware to ensure security on the generated MPSoC design. In this section, We provide a brief overview of SELinux's Flask, the MAC separation kernel we employed throughout this work.

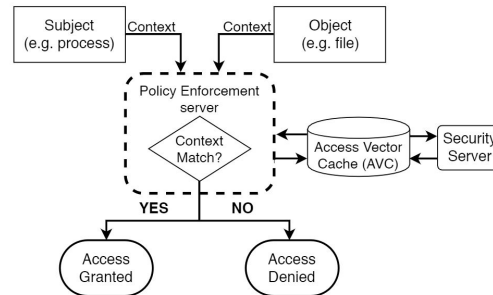


Figure 13.1: Flask security architecture. When a subject wants to perform an action on an object, it asks the policy enforcement server for permission. The policy manager knows about the objects it manages, and enforces the policy on behalf of the security server. The security server makes decisions based on the security policy. In order to not overwhelm the security server, AVC caches decisions and can grant or deny access on behalf of the security server.

To address the problem of domain separation enforcement in software environments, NSA, in conjunction with the Secure Computing Corporation and University of Utah, developed the open-source Flux Advanced Security Kernel (Flask) whose main objective is to provide flexible support for mandatory access control (MAC) policies in operating systems. In systems with MAC, a security label which describes the security context to which system components belong, is assigned to each system subject (e.g. processes) and each system object (files, sockets, memory segments, etc.). All accesses between subjects and objects must be governed by the MAC policy based on the labels. Flask cleanly separates the definition of the security policy logic from the enforcement mechanism, in order to enable different models of security to be enforced by the same base system. The security policy logic is designed and implemented in specialized language such as XML by the system administrator and is compiled into binary loadable to the separate component of the kernel, known as “security server”. Security policies enforcement is managed by “object managers”. The latter are regular kernel subsystems (e.g. process management, file-system, socket IPC, etc), modified with interfaces to obtain security policy decisions from the “security server”. To minimize the overhead associated with permissions lookup, Flask’s security server allows object managers to maintain an access

vector cache (AVC) component to store access decision computations which are provided by the security server for their subsequent usage. Figure \ref{fig:flask-image} illustrates the model.

Flask is implemented by modern OSes such as SELinux, a Linux security module (LSM) built into the kernel code. The default security policy it implements is the “Type Enforcement (TE)” policy. In TE, access is only granted between subjects and objects which share the same security context label “Type”.

To get detail on the architecture, users are encouraged to go through the publications on MeXT-SE. They are listed below:

### References:

1. Md Jubaer Hossain Pantho and Christophe Bobda. 2020. **MeXT-SE: A System-Level Design Tool to Transparently Generate Secure MPSoC**. 28th IEEE International Symposium on Field-Programmable Custom Computing Machines (**FCCM**), Fayetteville, AR, USA
2. F. Hategekimana, Joel Mandebi Mbongue, Md Jubaer Hossain Pantho, and Christophe Bobda. 2018. **Inheriting Software Security Policies Within HW IP Components**. 26th IEEE International Symposium on FieldProgrammable Custom Computing Machines (**FCCM**), Boulder, CO, USA
3. F. Hategekimana, Joel Mandebi Mbongue, Md Jubaer Hossain Pantho, and Christophe Bobda. 2018. **Secure Hardware Kernels Execution in CPU+FPGA Heterogeneous Cloud**. 2018 International Conference on FieldProgrammable Technology (**FPT**), Naha, Okinawa, Japan
4. F. Hategekimana, T. Whitaker, M. J. H. Pantho and C. Bobda. 2017. **Secure Integration of Non-Trusted IPs in SoCs**. Asian Hardware Oriented Security and Trust Symposium (**AsianHOST**)