**Interfaz ARMv7 y Cyclone III para Módulo de Hardware de Seguridad**

**Análisis de requerimientos y diseño preliminar**

**Clientes**

Rommel García Hernández

[rommelgh@gmail.com](mailto:rommelgh@gmail.com)

**Desarrollador**

Rommel García Hernández

[rommelgh@gmail.com](mailto:rommelgh@gmail.com)

Tabla de contenidos

1. **Requerimientos**
   1. **Vista general**
      1. **Propósito**

El propósito de este documento es describir el proyecto Interfaz ARMv7 y Cyclone III para Módulo Hardware de Seguridad. El objetivo de este proyecto es ofrecer una interfaz que se enfoca en eliminar la dificultad para la inyección de datos durante el desarrollo de un *Hardware Security Module* (HSM) sobre el FPGA Cyclone III. De tal forma, se reduce el tiempo de pruebas, con lo cual, el proyecto se centra en el desarrollo de un HSM eficiente.

* + 1. **Objetivos**
* Implementar una interfaz gráfica de usuario que permita la captura de mensajes y llaves para la ejecución de algoritmos criptográficos sobre Raspberry Pi 3b
* Obtener un sistema expansible para la verificación de un HSM
* Implementar en una tarjeta de desarrollo DE0 con Cyclone III una interfaz de comunicación con la Raspberry Pi 3b
* Implementar una versión inicial de un HSM sobre el FPGA Cyclone III
  1. **Requerimientos de sistema propuestos**
     1. **Vista general**

En general se requiere un sistema capaz de enviar estímulos a un diseño de HSM implementado sobre Cyclone III.

* + 1. **Requerimientos funcionales**

Esta sección contiene todos los requerimientos de software a un nivel de detalle suficiente para habilitar el diseño de un sistema para satisfacer tales requerimientos, además de habilitar las pruebas para comprobar que el sistema es capaz de satisfacer los requerimientos.

* **Funcionalidad de la Interfaz de usuario**

**Requerimiento funcional RFI100**

El sistema debe ser capaz de mantener una conexión con la Raspberry Pi 3b mientras dure la petición una operación sobre un mensaje

**Requerimiento funcional RFI101**

El sistema debe ser capaz de enviar una petición utilizan un botón en la interfaz de usuario.

**Requerimiento funcional RFI102**

El sistema debe mostrar la salida del algoritmo directamente en pantalla.

* **Funcionalidad en la Raspberry Pi**

**Requerimiento Funcional RFF100**

El usuario será capaz de enviar mensajes utilizando la interfaz gráfica o creando una conexión directamente en el procesador, la cual estará limitada a las necesidades del cliente y será el encargado de implementar cualquier otra interfaz.

**Requerimiento funcional RFR101**

la interfaz en FPGA será controlada únicamente por la Raspberry Pi 3b.

**Requerimiento funcional RFR102**

El sistema debe contar con una interfaz a bajo nivel para comunicarse como el Cyclone III en un lenguaje como C.

* **Funcionalidad en la tarjeta DE0**

**Requerimiento Funcional RFD100**

El sistema debe ser capaz de mantener la configuración de la FPGA aún después de que la tarjeta es apagada.

* + 1. **Requerimientos de usabilidad**

El sistema debe ser usado sin necesidad de realizar conexiones adicionales más que las necesarias para acceder a la interfaz gráfica del Raspberry Pi 3b. Debe existir ayuda y documentación completa para cada operación que puede realizar el sistema.

* + 1. **Requerimientos no funcionales**

**Requerimiento no funcional RNF100**

la aplicación debe tomar menos de 5 segundos en estar liast para ser usada.

**Requerimiento no funcional RNF101**

La aplicación debe ser intuitiva y tomar menos de 2 horas en ser manejada.

**Requerimiento no funcional RNF102**

la aplicación se entregará en un repositorio de GitHub.

**Requerimiento no funcional RNF103**

los lenguajes utilizados serán Python, C y Verilog.

**Requerimiento no funcional RNF104**

La tarjeta utilizada será la DE0 de Terasic con Cyclone III.

**Requerimiento no funcional RNF105**

La comunicación se implementará preferentemente con GPIO.

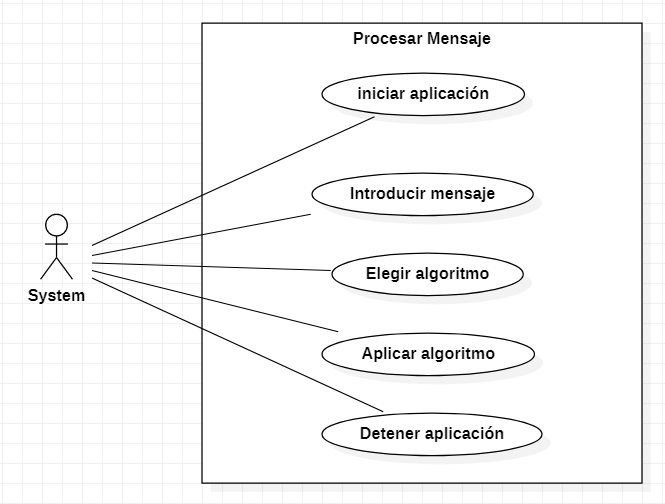
**Requerimiento no funcional RNF106**

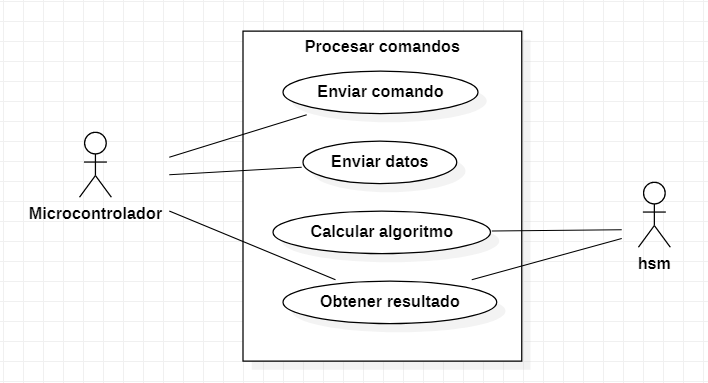
La aplicación debe ser extensible a utilizar archivos como entrada.

**Requerimiento no funcional RNF107**

los mensajes de menos de 1KB deben durar menos de un segundo en ser procesados.

* 1. **Modelos de requerimientos**
     1. **Diagramas de casos de uso**





* + 1. **Especificación de casos de uso**

|  |  |  |
| --- | --- | --- |
| **Caso de uso:** | **Iniciar la aplicación** | **CU100** |
| **Descripción**: El usuario hace una navegación a dónde el sistema fue descargado y ejecuta la aplicación app.py dentro de MVC. | | |
| **Actores**:Usuario, Sistema. | | |
| **Precondiciones**: La librería dinámica de la aplicación debe estar compilada o utilizar la librería que se descarga con el sistema. | | |
| **Flujo normal**:   1. El usuario navega hasta la ubicación de app.py dentro de <repo>/mvc/app.py 2. El usuario ejecuta la aplicación desde la terminal de Linux 3. El sistema carga las librerías necesarias 4. El sistema está listo para ser usado | | |
| **Flujo Alternativo**:   1. Si el usuario no tiene compiladas las librerías de C necesarias, deberá ejecutar el Makefile para obtenerlas | | |

|  |  |  |
| --- | --- | --- |
| **Caso de uso:** | **Introducir mensaje** | **CU101** |
| **Descripción**: El usuario escribe caracteres en una caja de texto de TkInter. | | |
| **Actores**:Usuario, Sistema. | | |
| **Precondiciones**: La librería dinámica de la aplicación debe estar compilada o utilizar la librería que se descarga con el sistema. | | |
| **Flujo normal**:   1. El usuario escribe un mensaje en una caja de texto de TkInter. El mensaje es una secuencia de caracteres | | |
| **Flujo Alternativo**:   1. El usuario debe introducir texto con la finalidad de poder utilizar las siguientes opciones del sistema | | |

|  |  |  |
| --- | --- | --- |
| **Caso de uso:** | **Aplicar algoritmo** | **CU101** |
| **Descripción**: El usuario presiona el botón calcular desde la interfaz gráfica de usuario. | | |
| **Actores**:Usuario, Sistema. | | |
| **Precondiciones**: El algoritmo está seleccionado y el mensaje introducido, además, si es necesario la llave está introducida. | | |
| **Flujo normal**:   1. El usuario tiene listos los datos de entrada del algoritmo 2. El usuario presiona el botón calcular 3. La Raspberry Pi 3b envía los datos de entrada del algoritmo por medio de la interfaz con el FPGA 4. los datos de entrada se almacenan en un buffer en la FPGA 5. FPGA identifica los datos del algoritmo y el algoritmo elegido 6. El FPGA ejecuta la máquina de estados del algoritmo 7. El FPGA envía el resultado del algoritmo e indica a la Raspberry Pi 3b que el dato está listo | | |
| **Flujo Alternativo**: | | |

|  |  |  |
| --- | --- | --- |
| **Caso de uso:** | **Enviar comando** | **CU101** |
| **Descripción**: El microcontrolador construye el código del algoritmo, los parámetros de entrada y los almacena en una región de memoria de trabajo. | | |
| **Actores**:Usuario, Sistema. | | |
| **Precondiciones**: El algoritmo está seleccionado y el mensaje introducido, además, si es necesario la llave está introducida. | | |
| **Flujo normal**:   1. El usuario presiona el botón calcular 2. El microcontrolador obtiene los datos de la aplicación y los ordena en el buffer de salida. 3. El microcontrolador agrega el comando que identifica al algoritmo a calcular 4. El microcontrolador usa DMA para enviar los datos por puerto serial. | | |
| **Flujo Alternativo**: | | |

|  |  |  |
| --- | --- | --- |
| **Caso de uso:** | Enviar datos | **CU101** |
| **Descripción**: el microcontrolador incia la comunicacion serial y el servicio de dma | | |
| **Actores**:Usuario, Sistema. | | |
| **Precondiciones**: los parametros de entrada están listos en la parte de modelo de la aplicación y el botón de calcular fue presionado. | | |
| **Flujo normal**:  el miconcontrolador contruye el buffer de salida  el microcontrolador agrefa la información del algoritmo  el micontrolador incia la comunicación serial | | |
| **Flujo Alternativo**: | | |

|  |  |  |
| --- | --- | --- |
| **Caso de uso:** | Calcular algoritmo | **CU101** |
| **Descripción**: el microcontrolador inicia la comunicación serial y el HSM (fpga) ejecuta el algoritmo sobre los datos enviados | | |
| **Actores**:Usuario, Sistema. | | |
| **Precondiciones**: los datos están en el buffer de entrada del HSM | | |
| **Flujo normal**:   1. La interfaz de recepción del FPGA construye el buffer de entrada 2. El control del HSM inicia la adquisición de datos del buffer de entrada y controla la ejecución del algoritmo 3. El control de la FPGA coloca el resultado en un buffer de salida 4. La interfaz del FPGA envía los datos por puerto serial | | |
| **Flujo Alternativo**: | | |

|  |  |  |
| --- | --- | --- |
| **Caso de uso:** | Obtener resultado | **CU101** |
| **Descripción**: el microcontrolador recibe los datos por puerto serial y la aplicación los presenta al usuario | | |
| **Actores**:Usuario, Sistema. | | |
| **Precondiciones**: Los datos están en el buffer de entrada del HSM | | |
| **Flujo normal**:   1. El controlador recibe la notificación de que hay nuevos datos en el bus de entrada 2. El microcontrolador envía los datos a la aplicación para que sean presentados   la vista de la aplicación presenta los resultados en la interfaz de usuario | | |
| **Flujo Alternativo**: | | |

1. **Diseño preliminar**
   1. **Interfaz de usuario**

La interfaz de usuario se implementa con la plataforma TkInter de Python, además, se utiliza el sistema operativo Raspbian para la interacción con la Raspberry Pi 3b.

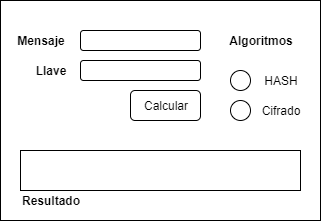


Figura 1. Prototipo de la Interfaz Gráfica de Usuario

* 1. **Diseño del sistema**
     1. **Diagrama de despliegue**

En esta sección se muestra la arquitectura del sistema en forma de un diagrama de despliegue y se identifican los componentes de esta.

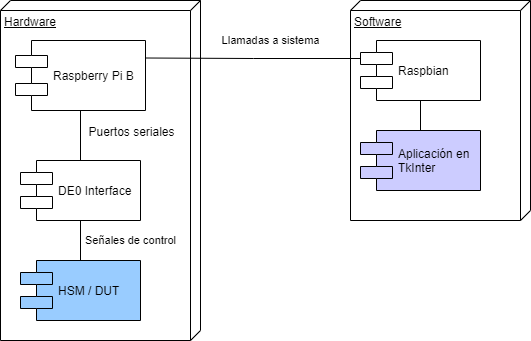


Figura 2. Diagrama de despliegue de la arquitectura

* + 1. **Modelo vista controlador**

Se implementa una interfaz gráfica de usuario utilizando el esquema de modelo, vista y controlador.

Por un lado, se implementa la vista con la librería TkInter de Python. En la cual, se presentan cuadros de texto, botones de checado y botones de acción.

El modelo de datos consiste en listas con la información proveniente de las entradas que provee la vista. Esta capa es implementada en Python.

En el caso del controlador, se implementa tanto en Python como en C/C++. En Python se implementa la lógica para manejar las transacciones solicitadas desde la interfaz de usuario y se conecta a otra parte del controlador que se desarrolla en C/C++. En esta última se implementa la lógica a bajo nivel para el control de los puertos seriales.

**Referencias**

[1] IEEE Guide for Software Requirements Specifications," in IEEE Std 830 - 984 , vol., no., pp.1-26, Feb. 10 1984, doi: 10.1109/IEEESTD.1984.119205,

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=278253&isnumber=6883