SimCardSniffer

Sistemas Embebidos – 2008

Universidad de la República

Facultad de Ingeniería

Integrantes:

Martín Alonzo

Andrés Bergeret

Juan Pablo González

Tutores:

Leo Steinfeld

Julio Pérez

Leo Etcheverry

Tabla de Contenido

Resumen....3

Introducción....3

- Descripción del problema....3
- Antecedentes....3

Objetivos....4

Alcance....4

Implementación Hardware....5

- Diseño del microprocesador NIOS II....5
- Conexión de los pines del SIM....7
- Validación del Hardware....8
- Implementación del Core de Detección de Frecuencia....9
- Configuración de la Placa IIE-Cyclone....10

Diseño del Software....11

- Propósito del sistema....11
- Objetivo del diseño....11
- Requerimientos de diseño....12
- Recepción de comandos y ejecución:....12
- Definición de la Arquitectura....13

Implementación Software....15

Pruebas....16

- i) Envío de comandos....16
- ii) Detección de cambios en CLK....16
- iii) Captura de datos por IO....16
- iv) Envío de eventos por la pc_uart....17
- v) Test de Integración....17

Conclusiones....18

Anexo: El ATR....19

Resumen

Se desarrolló un sniffer que registra el tráfico de datos entre un teléfono celular y su tarjeta SIM. Captura además eventos de cambio en la frecuencia de reloj y en las señales de Reset y alimentación del SIM, reportando toda la información hacia un PC.
Todo el sistema, formado por un microprocesador softcore NIOS II y algunos periféricos diseñados a medida, están embebidos dentro de un FPGA en una configuración de System on Chip (SoC).

Introducción

Descripción del problema

El proyecto consistió en configurar un microprocesador softcore NIOS II en una plataforma de desarrollo que cuenta con un FPGA de la empresa Altera y programar en él un sniffer de comunicación y otras señales entre un teléfono celular y su tarjeta SIM, así como implementar el envío de estos datos a un PC.

Antecedentes

- Este proyecto tiene como marco un convenio entre Facultad de Ingeniería y ANTEL, en el cuál se busca asesorar al ente en la seguridad de los distintos servicios que el mismo presta. Antel buscaba disponer de una herramienta para capturar los detalles de la comunicación entre el teléfono celular GSM y el SIM.
- La placa IIE-Cyclone a ser usada en el Proyecto, fue desarrollada como proyecto de grado del IIE bajo la tutoría de J. P. Oliver (2007)
- Como proyecto del curso Diseño Lógico 2 de 2007 se implementó una UART modificada para comunicación de tarjetas SIM

Objetivos

- Desarrollo de un software embebido que permita la captura de las señales entre un teléfono GSM y su tarjeta SIM, y el envío de la interpretación de las mismas a un PC.
- Implementación del hardware necesario para el correcto funcionamiento del software desarrollado.

Alcance

i. Configuración, en el FPGA Cyclone II de la placa IIE-Cyclone, del microprocesador embebido NIOS II de la empresa Altera y adaptación del ambiente de desarrollo del mismo para dicha plataforma.

ii. Desarrollo de un “IP Core”, en VHDL, para la detección de frecuencia (y nivel de tensión alto o bajo, en caso de estar detenido) de la señal de reloj del SIM.

iii. Implementación hardware para capturar en la placa IIE-Cyclone las señales entre el teléfono y su tarjeta SIM.

iv. Programación del microprocesador para cumplir las siguientes funciones: registro de eventos con timestamp y envío de los mismos a un PC para su procesamiento posterior.
Los eventos a registrar son los siguientes:

- Cambios de nivel en los pines de alimentación y reset
- Datos transmitidos
- Desborde en el contador utilizado para agregar el timestamp

Tareas comprendidas dentro del Alcance inicial y posteriormente excluidas del mismo:

v. Realización de un core IP de una UART modificada para captura del protocolo de comunicación del SIM. La modificación consiste en detectar la situación en que el receptor solicita retransmisión del dato, y hacer eso disponible como una bandera de status accesible desde el programa.

vi. Recepción desde el PC de comandos para inyectar distintas fallas en la comunicación.

Implementación Hardware

Diseño del microprocesador NIOS II

Debido a que utilizamos un microprocesador NIOS II, el cuál es un microprocesador “softcore”, tuvimos la posibilidad de configurar diversos parámetros del mismo.
Por restricciones del FPGA en el que lo “embebimos”, optamos por un diseño mínimo para cumplir con los requerimientos de la aplicación.
Es importante señalar que la utilización de un microprocesador de esta naturaleza permite una gran flexibilidad y la posibilidad de utilizar un micro orientado a la aplicación, evitando quedar muy restringido en las capacidades del mismo, o, por el contrario, usar un micro sobredimensionado.
Por contrapartida, el proceso de diseño del mismo insume mucho tiempo, dado que es un proceso iterativo en el que se cuenta con muchos grados de libertad, lo que nos exige realizar diversas pruebas funcionales.

A continuación, presentamos los distintos cores que fueron creados con el SOPC Builder, la herramienta del Quartus II que permite la creación de componentes VHDL con interfaz de usuario gráfica.

a) Procesador Nios II:

Utilizamos un Nios II/s: un microprocesador de 32 bits del tipo RISC, con algunas funcionalidades adicionales como Cache de Instrucciones y multiplicación y división por hardware. Lo conectamos a la señal del oscilador de la placa IIE-Cyclone, de 25 MHz a través del core sys_clk_timer.

b) Memoria On-Chip:

Utilizamos la máxima memoria on-chip permitida por nuestro FPGA: 20 Kbytes para memoria de programación y de datos. Utilizamos memoria RAM de 32 bits de ancho de palabra.

c) PIOs:

Utilizamos diversos PIOs dependiendo de su aplicación.
En los casos en que correspondieran a entradas que queríamos sensar, utilizamos PIOs con la capacidad de generar interrupciones por flancos de subida o bajada (vs_pio para sensar las señales de RESET y de VCC del SIM y freq_count_pio para la entrada del bloque de detección de frecuencia.)
Usamos PIOs sin interrupciones para generar salidas (seven_seg_pio y led_pio) o para sensar entradas por polling (dip_switch_pio.)

d) UARTs:

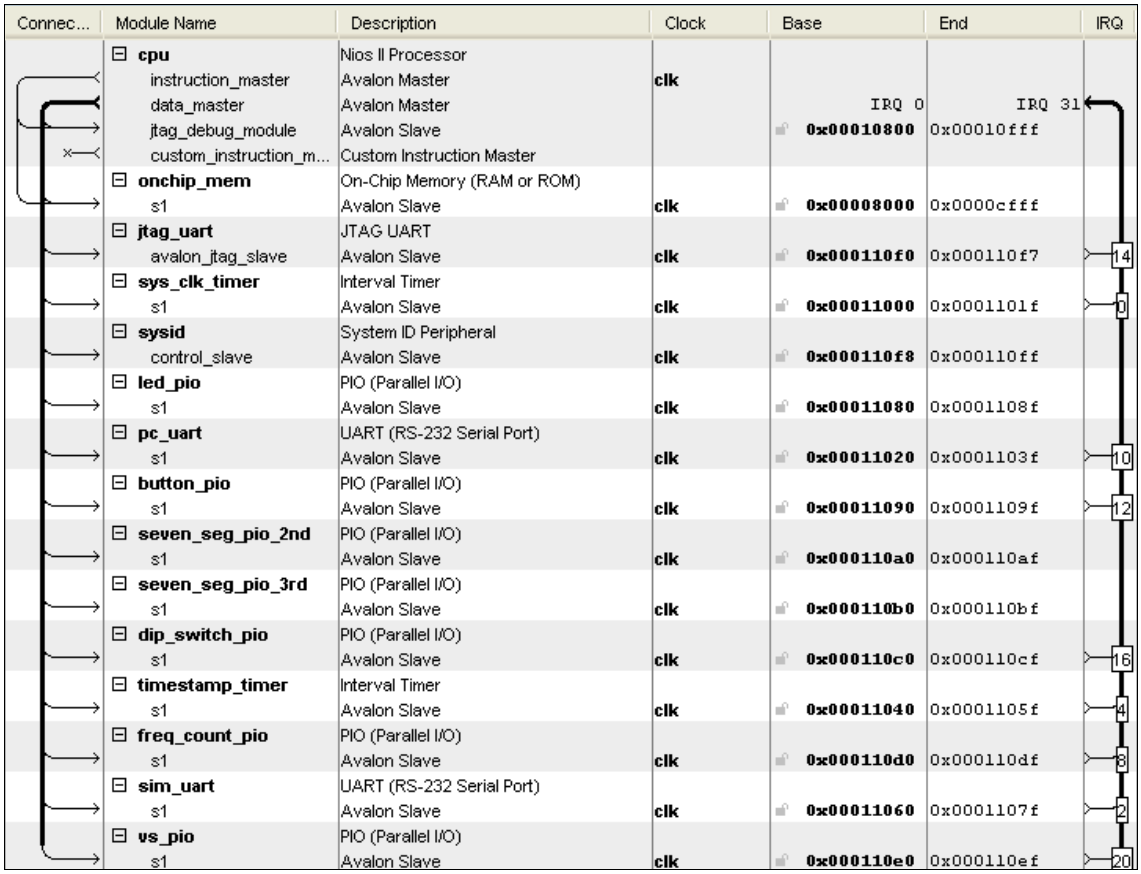
Utilizamos dos UARTs: una para “sniffear” la comunicación entre el teléfono y su tarjeta SIM (sim_uart) y otra para la recepción de comandos y el envío de datos al PC (pc_uart).
La pc_uart utiliza una velocidad fija (28800 bps), 8 bits de datos y 1 de parada.

La sim_uart, mientras tanto, utiliza 8 bits de datos, 1 bit de parada y 1 bit de paridad par y velocidad configurable por software, de manera de poder alterarla de acuerdo a cambios en la frecuencia utilizada por la SIM o por una gestión de cambio de velocidad a petición del teléfono (en el ATR.)

e) Otros:

Otros componentes fueron utilizados para las pruebas: timestamp_timer para generar sellos de tiempo (en la versión final se temporizó con herramientas de software) y la jtag_uart para comunicación con la PC. También se utilizó el core sysid por sugerencia de la documentación del Nios II.

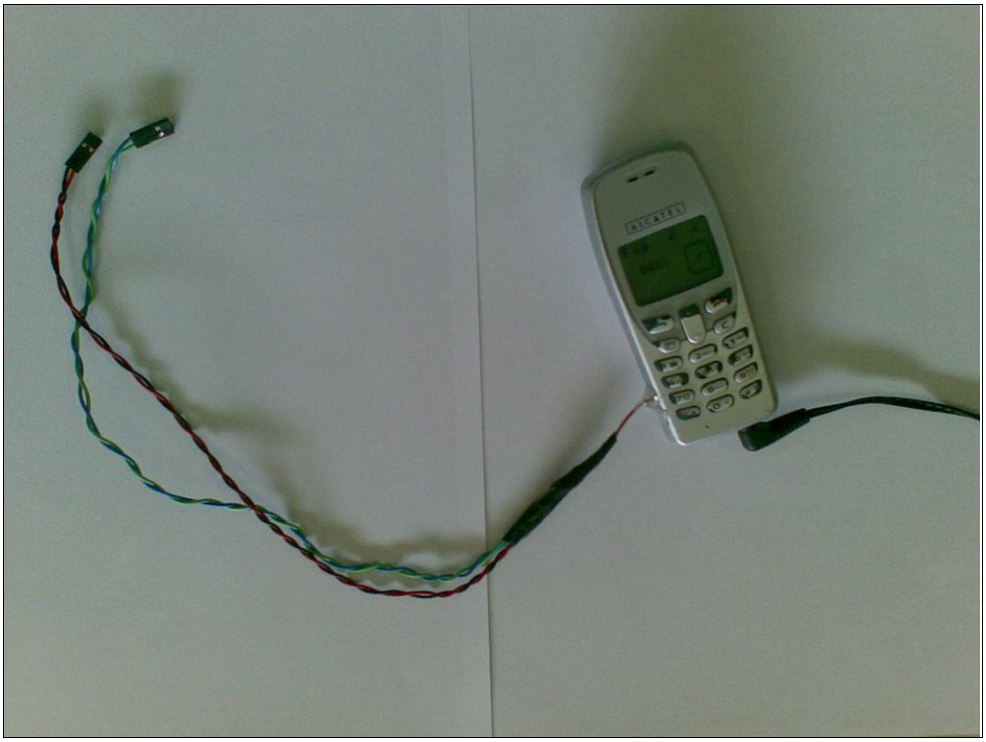
En la imagen se muestra una captura de pantalla del SOPC Builder con todos los componentes utilizados:



Connec...	Module Name	Description	Clock	Base	End	IRQ
	cpu	Nios II Processor				
	instruction_master	Avalon Master	**clk**			
	data_master	Avalon Master			IRQ 0	IRQ 31
	jtag_debug_module	Avalon Slave		**0x00010800**	0x00010fff	
	custom_instruction_m...	Custom Instruction Master				
	onchip_mem	On-Chip Memory (RAM or ROM)				
	s1	Avalon Slave	**clk**	**0x00008000**	0x0000cfff	
	jtag_uart	JTAG UART				
	avalon_jtag_slave	Avalon Slave	**clk**	**0x000110f0**	0x000110f7	14
	sys_clk_timer	Interval Timer				
	s1	Avalon Slave	**clk**	**0x00011000**	0x0001101f	0
	sysid	System ID Peripheral				
	control_slave	Avalon Slave	**clk**	**0x000110f8**	0x000110ff	
	led_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**clk**	**0x00011080**	0x0001108f	
	pc_uart	UART (RS-232 Serial Port)				
	s1	Avalon Slave	**clk**	**0x00011020**	0x0001103f	10
	button_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**clk**	**0x00011090**	0x0001109f	12
	seven_seg_pio_2nd	PIO (Parallel I/O)				
	s1	Avalon Slave	**clk**	**0x000110a0**	0x000110af	
	seven_seg_pio_3rd	PIO (Parallel I/O)				
	s1	Avalon Slave	**clk**	**0x000110b0**	0x000110bf	
	dip_switch_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**clk**	**0x000110c0**	0x000110cf	16
	timestamp_timer	Interval Timer				
	s1	Avalon Slave	**clk**	**0x00011040**	0x0001105f	4
	freq_count_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**clk**	**0x000110d0**	0x000110df	8
	sim_uart	UART (RS-232 Serial Port)				
	s1	Avalon Slave	**clk**	**0x00011060**	0x0001107f	2
	vs_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**clk**	**0x000110e0**	0x000110ef	20

Conexión de los pines del SIM

Para poder hacer la conexión física entre las señales de la tarjeta sim y la placa IIE-Cyclone en la que reside el microprocesador, soldamos cables en el zócalo de tarjeta SIM de un teléfono celular GSM (Alcatel One Touch 320).
En los extremos de los cables pusimos conectores hembra de 2.54mm (2 conectores de 2x1) como se observa en la imagen. Luego separamos los cables de RESET y GND para facilitar su conexión a la placa.



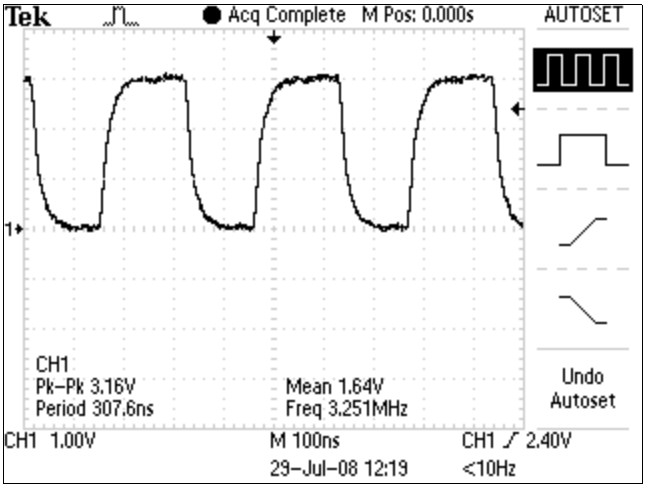
El conexionado al expansor EXP1 de la placa IIE-Cyclone que realizamos para las pruebas fue el indicado en la tabla:

Cable	**Señal**	**Pin EXP1**	**Pin FPGA**
Negro	GND	4	GND
Rojo	RESET	31	13
Azul	CLK	36	8
Verde	DAT_IO	35	7

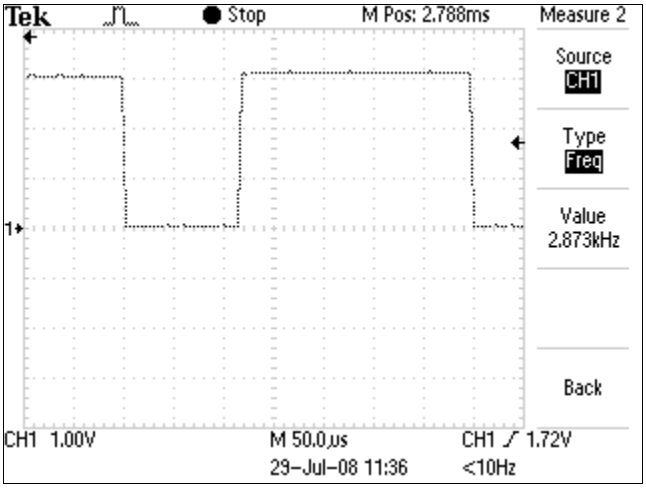
Validación del Hardware

Para probar la correcta recepción de las señales del SIM, hicimos varias pruebas con un osciloscopio digital, en las que obtuvimos resultados coherentes con lo establecido en la especificación técnica GSM 11.11, lo que nos permitió confirmar lo hecho y seguir avanzando con las demás etapas del desarrollo.

- Voltaje de alimentación del SIM: 3.05 VDC
- Señal de RESET: Activa por bajo a 3.05 V
- Frecuencia del reloj del SIM (f_CLK): 3.25 Mhz



- Tiempo de bit (T_bit) de la señal de datos DAT_IO: 11.67 us



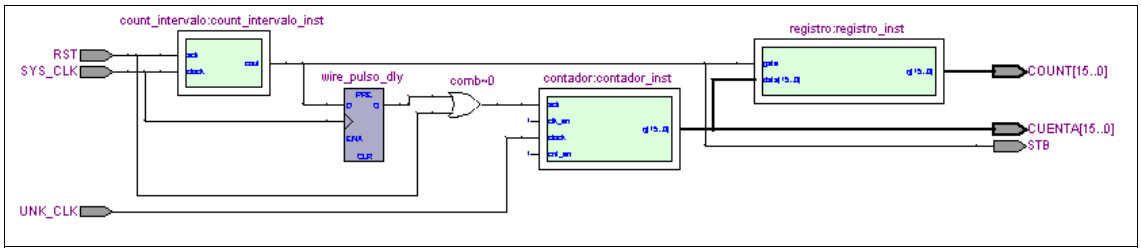
- f_CLK * T_bit = 379 (muy cercano a los 372 que establece GSM 11.11)

Implementación del Core de Detección de Frecuencia

Observando que el rango de frecuencias posibles llegaba hasta los 10 MHz (y en el teléfono utilizado era de 3.25 MHz) decidimos que no era eficiente una detección de frecuencia por software, por lo que se decidió hacer un Core VHDL de detección de frecuencia.

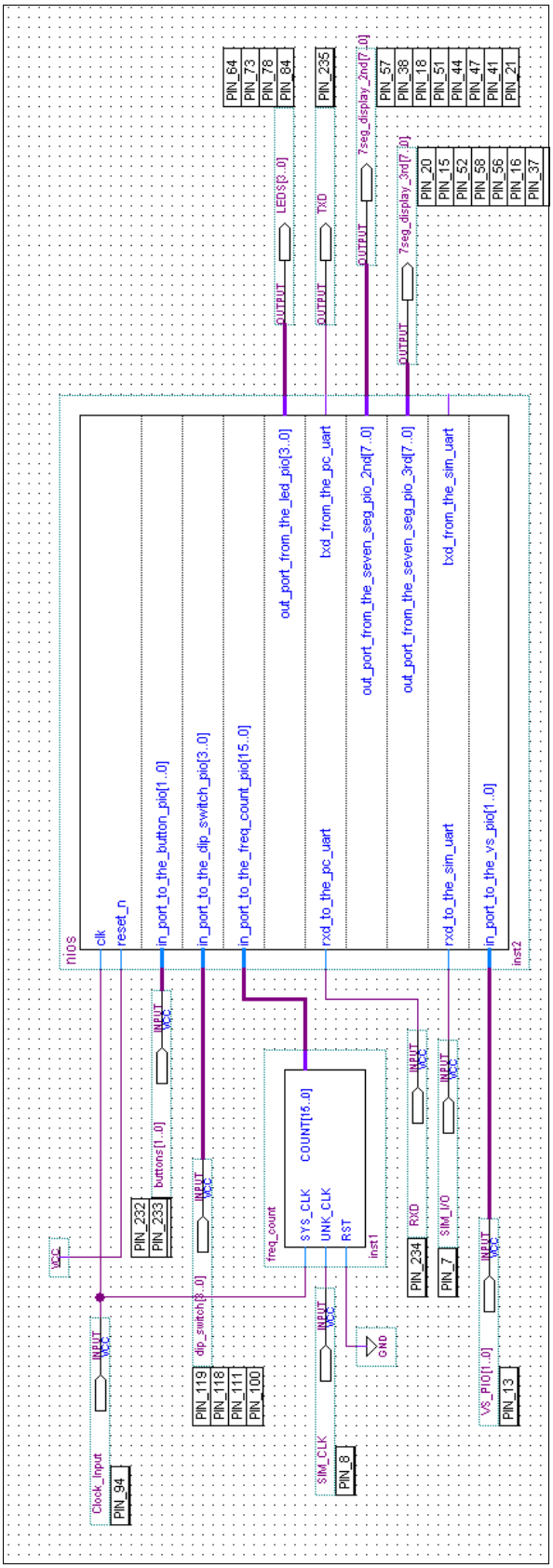
El mismo cuenta los flancos de reloj en un intervalo de tiempo conocido (2 ms) y envía el resultado de la cuenta a través del freq_count_pio.

La figura muestra un diagrama de bloques del core completo que incluye un contador de constante conocida para generar el intervalo de tiempo y otro contador para los flancos del reloj del SIM (UNK_CLK). El reloj del diseño (SYS_CLK) es de 50 MHz (el doble de la frecuencia del oscilador, lo que se logra con un core PLL x2,.)



Configuración de la Placa IIE-Cyclone

En la figura se puede observar el diagrama de bloques del diseño completo con la asignación de pines incluida.



Diseño del Software

Propósito del sistema

Sistema para captura de datos y generación de eventos en la comunicación SIM – TE.
El sistema está conectado al bus de señales entre SIM-TE, estos corresponden a las señales CLK (reloj del SIM proporcionado por el TE), I/O (señal de datos bidireccional, el protocolo arbitra), RST (señal de reset), Vcc (voltaje del pin de alimentación del SIM.)
Los eventos se deben generar como consecuencia de variaciones determinadas de estas señales. Los eventos estarán caracterizados por los valores de determinadas propiedades de esas señales y una marca de tiempo que será relativa a un instante a determinar y se representa el instante en el que ocurrió el evento.

El sistema debe capturar todos los eventos definidos en el documento de diseño durante el tiempo en que la generación de eventos esté habilitada. Se podría contar con la posibilidad de seleccionar qué eventos se desea capturar durante la ejecución, habilitando o deshabilitando subconjuntos de éstos.
El sistema debe poder conectarse a un terminal externo que permita la operación en tiempo real, y el volcado de eventos del sistema.
El sistema procurará no desechar eventos por estar imposibilitado de registrarlos, por ejemplo por carecer de memoria. Para esto, el sistema debe vaciar sus colas de eventos cuando se queda sin espacio para registrar más eventos, a través de la interfaz prevista para la operación externa.

Debe ser posible en un desarrollo posterior, la inyección de fallas a la comunicación, y el seguimiento del protocolo comunicación. Esto permitiría alterar físicamente la comunicación de forma controlada abriendo la posibilidad de realizar un conjunto de experimentos más extenso que el soportado con la versión desarrollada, acerca de la seguridad los dispositivos SIM.
La interpretación del protocolo de comunicación permite generar eventos asociados a parámetros de nivel de aplicación, además permite generar complejas condiciones de captura de datos e inyección de fallas, por ejemplo, capturar “la respuesta a la introducción del numero de PIN”, o inyectar una falla en la línea de comunicación durante el n-ésimo byte de la respuesta a determinado comando solicitado por el TE al SIM.

Objetivo del diseño

Dada la flexibilidad del hardware utilizado para el sistema, el diseño de software no será independiente del del hardware, las decisiones de realizar parte de una determinada funcionalidad en SW o HW se tomarán en conjunto. Por tanto, el objetivo del diseño consiste en definir una arquitectura de software y especificar las adaptaciones necesarias al hardware de manera que la implementación de la arquitectura cumpla con el propósito del sistema.
En esta etapa del proyecto, el HW está en parte seleccionado, y en parte está pendiente de la definición de los dispositivos periféricos que debe contar para que el sistema cumpla su propósito.

La selección del Nios II softcore, y la placa IIE-Cyclone fue propuesta por el equipo docente del curso con el objetivo de fomentar el aprendizaje en el desarrollo de sistemas usando softcores y aplicación de la placa desarrollada en el Instituto de Ingeniería Eléctrica.

Requerimientos de diseño

Para el diseño del SW se describe una lista de requerimientos que el sistema debe cumplir

Detectar y Registrar los siguientes eventos:

- IO_EVENT: transmisión de un byte por la línea IO. El evento consiste en la captura del dato transmitido junto con un timestamp.
- FREQ_EVENT: es generado al detectar un cambio en la frecuencia de CLK cuando este cambio es superior a cierto umbral preestablecido. El evento consiste en la medida de la frecuencia de CLK, y el timestamp.
- VS_EVENT: al cambiar alguno de los pines Vcc o RST, se debe registrar el evento, conteniendo como parámetros la indicación de los pines que cambiaron, el nuevo valor de los mismos y el timestamp.

Los distintos tipos de eventos imponen límites en los tiempos de respuesta de los segmentos de código encargados de registrar el evento como respuesta.

IO_EVENT puede ocurrir cada un tiempo de byte de IO, esto es a su vez, el tiempo correspondiente a 10 (10 tiempos de bit por cada byte trasmitido) veces 372 periodos de reloj de CLK, siendo el reloj como máximo 5 MHz, lo que da un tiempo máximo de atención de 744 us.

FREQ_EVENT podrá generarse a lo sumo con una cadencia de 500 por segundo, o sea cada 2 ms. Este tiempo se obtuvo de los parámetros de implementación del detector de frecuencia que fue desarrollado en HW. En los criterios de diseño HW se justifica esta cadencia.

VS_EVENT no se puede especificar de antemano la frecuencia con que ocurrirá este evento, de la especificación del protocolo se puede extraer los tiempos mínimos de cambio de RST antes del ATR. Aquí el diseño será buscar el menor tiempo posible de atención a esta rutina para no afectar las respuestas de las otras, pero sin garantizar tiempos mínimos para la respuesta.

Recepción de comandos y ejecución:

Los comandos se deben recibir desde un sistema externo por medio de algún protocolo de comunicación sencillo que permita interacción con el usuario en tiempo real.

Estas tareas de comunicación también ocuparán tiempo de ejecución del CPU. Para minimizar el impacto de éstas en la duración de las rutinas de atención a la generación de eventos, se debe escoger una arquitectura que dé prioridad a

la generación de eventos frente a la comunicación de gestión y otras tareas generadas internamente.

Definición de la Arquitectura

Para la definición de la arquitectura software se harán los siguientes supuestos:

- Se cuenta con un procesador Nios II con un reloj de sistema de 25 MHz y 32 KB de memoria para programa y datos.
- Se conocen los tiempos de ejecución de las instrucciones del Nios II de la documentación
- Se tienen restricciones de tiempos de procesamiento de los eventos

Con el objetivo de tener rutinas, de atención suficientemente rápidas y optimizar el uso el tiempo de la CPU, y paralelamente implementar mecanismos de planificación de tareas más complejos que el polling, tan solo con fines didácticos ya que el sistema era igualmente viable con estructuras simples, se propone una arquitectura basada en interrupciones y planificación por cola de tareas.

Los eventos que se desea registrar están vinculados directamente a cambios en las señales de la interfaz entre el SIM y el TE, por tanto se puede conectar estos pines a los periféricos del HW y programarlos para que interrumpan al detectar esos cambios.
Para la captura de datos se propuso el uso de una UART del Nios II que estaría conectada con su entrada Rx a la línea de datos de la interfaz SIM-TE. La UART debe entonces generar una interrupción al detectar alguno de los siguientes eventos: nuevos datos recibios completamente, error de paridad, error en bit de parada, error de entramado. La ISR deberá interpretar la causa de la interrupción y registrar el evento correspondiente. Adicionalmente, si se desea tener seguimiento de la comunicación, en la ISR se podría invocar a la operación “update” de la maquina de estados que cumple esa función o pasar la actualización como una tarea mas al hilo de ejecución central, dependiendo de la velocidad de respuesta y prioridad requerida. Para este proyecto solo se implementara la interrupción causada al tener datos disponibles para leer de la UART.

Para el sensado de las señales Vcc y RST, se propone la conexión a un PIO asignando un pin a cada señal y configurándolo con interrupciones sensibles a flanco de subida y de bajada. De esta forma se detecta el cambio de alguna de estas señales y se puede leer el nuevo valor.

La detección de frecuencia de CLK debe ser lo suficientemente rápida para detectar los cambios de frecuencia con la precisión especificada. Para evitar que el CPU se encargue de medir la frecuencia para detectar si cambió, se genero el módulo de HW de detección de frecuencia. Como se describe en la sección de diseño de HW, el detector de frecuencia presenta la cuenta de pulsos de CLK en un período específico de tiempo. Este valor será utilizado como señal de entrada de un PIO que interrumpe sensible al flanco de subida o de bajada de sus pines de entrada. El SW se encarga de calcular la frecuencia

y comparar con el valor último medido de frecuencia y generar un evento sólo si el nuevo cálculo de frecuencia supera un cierto nivel de tolerancia a cambios en las medidas consecutivas.

La recepción de comandos y envío de respuestas se basará en la programación de los drivers específicos para la atención de interrupciones de recepción de datos y de envío de datos listo de la UART creada para ese propósito.

La gestión de eventos debe permitir crear un evento en el ambiente de las ISR, guardarlo en memoria y conservarlo hasta que se decida enviar al exterior del sistema. Este gestor de eventos debe proporcionar una interfaz con tal objetivo y debe administrar la memoria del sistema de forma dinámica.

La planificación de tareas por medio de cola, se hace en el hilo de ejecución central. Para ello se dispone de una estructura FIFO circular donde las ISR van registrando tareas que son punteros a funciones definidas e implementadas en el sistema. En el hilo principal se está pendiente de la aparición de una nueva tarea para ejecutar en un ciclo repetitivo. Las tareas se van extrayendo en el orden que se colocaron y se ejecutan.

Implementación Software

La implementación del software se realizó siguiendo las directivas del documento de diseño SW utilizando el entorno de desarrollo Nios II EDS.
Para cada uno de los periféricos se implementó un módulo que incluye las funciones de inicialización “scs_init_xxxx”, las ISR y funciones necesarias para la funcionalidad. Para el registro de las ISR se utilizaron las funciones “alt_irq_register” y para la habilitación y deshabilitación las funciones “alt_irq_enable” y “alt_irq_disable” respectivamente.
El acceso a los registros de Status, Control y otros (dependiendo de cada periférico en particular) se realizó utilizando las macros provistas por el HAL del Nios II generado por el EDS, a saber, IORD_<periferico>_<registro>(base), IOWR__<periferico>_<registro>(base , valor) y las constantes de definición de los bits de cada registro.
Estas macros y constantes se definen para cada periférico en los archivos “<periferico>_regs.h”. Esto se detalla en la documentación generada en el código usando doxygen.

Para la gestión de eventos se implementó un módulo “scs_events” que administra una cola de eventos para cada tipo de evento de los descriptos en el documento de diseño.
Estas colas FIFO circulares “scs_xxx_queue”, dependen de cada tipo de evento. El módulo de gestión de eventos proporciona funciones píblicas para el registro de eventos y tiene acceso al módulo generador de timestamps para estampar los eventos. El registro de cada evento se realiza en el contexto de las ISR. Además este módulo proporciona funciones para convertir los bytes de cada evento a un formato imprimible en ASCII. Estas funciones son llamadas al momento de enviar los datos al PC.

La generación de timestamps se realiza en el módulo “scs_timer”. Se utilizaron las funciones “alt_nticks” proporcionadas por la API de Nios II. El módulo provee funciones para inicializar la cuenta y para obtener un timestamp de 32 bits con la cuenta en milisegundos desde que se inicializó el contador.

La recepción de comandos desde el PC y el envío de datos desde el sniffer se implementó en el módulo “scs_pc_uart”. Este módulo administra los buffers de bytes para la recepción de los comandos y maneja el acceso compartido del buffer de transmisión. El problema de datos compartidos se resuelve habilitando y deshabilitando interrupciones. En el documento de diseño y arquitectura se especifica con más detalle el funcionamiento del módulo.

La planificación de tareas se realiza en el hilo principal de ejecución “scs_main” por medio de una cola de punteros a funciones (las tareas) que es continuamente consultada por la existencia de nuevos elementos para ejecutar. Cada ISR tiene acceso a esa cola para insertar tareas. Nuevamente se debió controlar el acceso a la cola de tareas ya que es compartida por las ISR y el hilo principal. Para esto se deshabilitan y habilitan las interrupciones antes de cada lectura de estado de la cola y antes de extraer un elemento para ejecutar.

Pruebas

Para la verificación de las funcionalidades del sistema se elaboraron las siguientes pruebas:

i. Envío de comandos simples para testear el módulo de recepción de comandos
ii. Detección de cambios de frecuencia de CLK
iii. Captura de datos por IO
iv. Envío de eventos por la UART hacia el PC
v. Test de integración

i) Envío de comandos

En esta prueba se testeó el módulo de recepción de comandos. El objetivo era mandar un comando simple que fuese interpretado en el sistema y desencadenera una acción visible desde el exterior. Para ello se implementó un comando que enciende los 4 LEDs y otro que los apaga.
Esta prueba requiere tener implementado el módulo de recepción de comandos con la pc_uart y el led_pio.
Los comandos son secuencias de bytes predefinidas, a saber: “led_on\n” y “led_off\n” para encender y apagar respectivamente.
Los resultados fueron satisfactorios pudiendo prender y apagar los 4 LEDs en función de la orden que se enviara desde el PC.

ii) Detección de cambios en CLK

Se probó el funcionamiento del frecuencímetro y de la lógica de cálculo de frecuencia a partir de los datos proporcionados por dicho periférico.
Los requerimientos de la prueba son: tener implementado el módulo del frecuencímetro HW y el SW, tener HW para conectar el CLK a la placa de desarrollo, tener mecanismo de salida de datos (se usó el jtag_uart).
La prueba consiste en iniciar el sistema y encender el celular durante algunos segundos. Durante el transcurso de la prueba se deberían ir registrando los cambios en la frecuencia y estos se deberían enviar hacia un terminal que los pueda mostrar.
Los resultados de las pruebas se contrastaron con las medidas realizadas en laboratorio utilizando un osciloscopio.
Luego de corregidos algunos errores en el sistema, detectados por la realización de estas pruebas, los resultados fueron coherentes con lo medido con el osciloscopio.

iii) Captura de datos por IO

La prueba consiste en capturar los datos que se trasmiten por IO entre SIM y TE para validar el correcto funcionamiento del módulo SIM – TE.
Como requisitos se tiene: implementación del módulo sim_uart para la captura de datos de IO, mecanismo para transmitir datos a un terminal (se usó pc_uart

ya verificada), HW para conexión del bus SIM-TE a la placa de desarrollo con el pin IO conectado.
Se supuso para la prueba (justificado en parte por el análisis empírico del comportamiento de la comunicación) que el baudrate de IO sería constante y que CLK tendría un valor fijo durante el transcurso de la prueba.
El procedimiento consistió en iniciar el sistema y a continuación encender el celular. Los datos transmitidos por IO se fueron registrando y a la vez enviando al terminal para que el usuario pudiese verlos y luego comparar con lo esperado.
Para la validación de los datos se utilizó el documento de descripción del protocolo de comunicación SIM-TE y los resultados de las mediciones con osciloscopio realizadas anteriormente.
Los resultados, luego de la depuración de errores, fueron satisfactorios. Se pudo comparar la primer ráfaga de datos (ATR) con lo esperado por el documento del protocolo y con lo medido en laboratorio con el osciloscopio habiendo coincidencia.

iv) Envío de eventos por la pc_uart

Con esta prueba se busca validar el funcionamiento de la pc_uart en transmisión de datos y las funciones de transformación de los eventos a ASCII.
Luego de tener varios eventos capturados, se da la orden de enviarlos por el pc_uart. Para enviar la orden se implementó un comando “get_events\n” que inicia el envío. Los eventos generados son: cambio de frecuencia de CLK y captura de datos por IO.
Como requerimiento se debe tener el módulo de gestión de eventos implementado y verificado, el módulo pc_uart implementado y una conexión con un terminal a través de la UART.
Para el desarrollo de la prueba se inicia el sistema en modo debug en el Nios II EDS, se inicia el celular y al cabo de algunos segundos se envía el comando “get_events”. Luego se compara la salida con los datos que registró el evento.
Los resultados, luego de la depuración, fueron exitosos.

v) Test de Integración

Consiste en mecanismos que permitan verificar el funcionamiento del sistema en su conjunto en un escenario de uso “real”.
Para ello se debe tener la implementación completa del sistema
Durante este test hay funcionalidades que no se pudieron ensayar por no ocurrir en la ejecución de la prueba. Este fue el caso de la re negociación del baudrate entre SIM y TE y ajuste de la sim_uart ante ese evento.
Como medida del éxito de las pruebas se usó el documento del protocolo y se comparó los resultados con lo que se esperaría teóricamente.
Los resultados fueron exitosos en la mayoría de los casos, quedando pendientes más pruebas de este tipo.

Conclusiones

- Se implementaron en forma exitosa los aspectos clave del sistema tal como se planificó en el diseño.
- Se logró obtener una arquitectura elegante para el manejo de prioridades basada en interrupciones y planificación por cola de tareas.
- Se obtuvo un sistema que puede crecer con facilidad en funcionalidades.
- Se pudo validar el funcionamiento correcto del sistema en una amplia cantidad de casos.
- El sistema está apto para implementar seguimiento del protocolo a más alto nivel.
- Falta realizar más pruebas en cuanto a la temporización de las ISR.
- El uso de Nios II facilitó la utilización de periféricos customizados para la aplicación.
- Queda abierta la posibilidad de dotar al sistema de un inyector de fallas dirigido por comandos o como respuesta a algún evento.
- La complejidad del sistema en desarrollo de HW y SW dificultó estimar los recursos necesarios para su desarrollo.

Anexo: El ATR

Después de enviar la información sobre voltaje de alimentación, reloj y señal de reset, la SIM manda un mensaje Answer to Reset (ATR) por la señal de I/O. Esta secuencia de datos, la cual contiene a lo sumo 33 bytes es siempre enviada con un valor de división (factor de conversión de la frecuencia de reloj) de 372, de acuerdo con el estándar ISO/IEC 7816-3. Este contiene varios parámetros relacionados con el protocolo de transmisión y la tarjeta. Este valor de división debe de ser usado inclusive si el protocolo de transmisión usado después del ATR utiliza un valor diferente (por ej, 64). Esto asegura que un ATR puede ser recibido desde cualquier tarjeta, sin importar los parámetros del protocolo de transmisión usados anteriormente.
Es muy raro que el ATR utilice el largo máximo disponible. Generalmente consiste en unos pocos bytes (en nuestro caso, alrededor de 15 a 20). Particularmente en aplicaciones donde la tarjeta debe ser usada rápidamente después de la secuencia de activación, el ATR debe ser corto. Un ejemplo típico es al pagar un peaje, donde tarjetas inteligentes son usadas, y se debe usar el mínimo tiempo posible.

El comienzo de la transmisión de ATR debe ocurrir entre 400 y 400000 ciclos de reloj, después de que la terminal quita la señal de reset.
Con una frecuencia de 3.5712 MHz, esto corresponde a un intervalo de 112 μs a 11.20 ms, mientras que 4.9152 MHZ, el intervalo es de 81.38 μs a 8.14 ms. Si el terminal no recibe el comienzo del ATR en este intervalo, este repite la secuencia de activación varias veces (usualmente mas de 3) para tratar de detectar un ATR. Si todos estos intentos fracasan, el terminal asume que la tarjeta es defectuosa y reacciona de acuerdo a eso.
Durante un ATR, el tiempo entre flancos de subida de 2 bytes sucesivos puede ser mayor que 9600 etu, de acuerdo con la ISO/IEC 7816-3.
Este período es designado como el “tiempo de espera inicial” y con un reloj de 3.5712 MHz, corresponde a exactamente un segundo.
Esto significa que el estándar permite un retardo de un segundo entre bytes individuales de un ATR, cuando es enviado al terminal.

En algunos sistemas operativos de las tarjetas inteligentes, este tiempo es utilizado para operaciones internas y accesos a escritura de la EEPROM. El buffer de escritura interior para operaciones atómicas a menudo es refrescado en el mismo intervalo de tiempo.

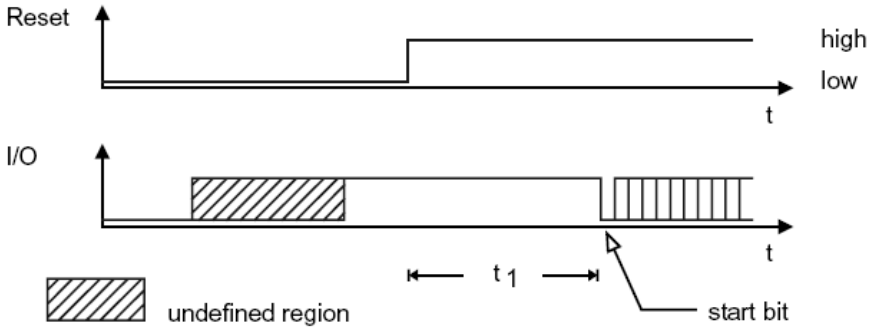


Figure 6.10 Timing diagram of the reset signal and the start of the ATR, in accordance with ISO/IEC 7816-3 (400 clock cycles $\leq t_1 \leq$ 40,000 clock cycles)

Los elementos y cadenas de datos del ATR están definidos en detalle en la norma ISO/IEC 7816-3.
El formato básico se ve en la siguiente figura y tabla. Los primeros 2 bytes, designados TS y T0, definen varios parámetros fundamentales de transmisión e indican la presencia de subsiguientes bytes.

La interfaz de caracteres especifica parámetros de transmisión especiales para el protocolo, los cuales son importantes para los siguientes datos trasmitidos. Los caracteres históricos describen la medida de las funciones básicas de la tarjeta inteligente. El carácter de chequeo, el cual es un chequeo de suma de los bytes previos, puede opcionalmente ser enviado en el último byte del ATR, dependiendo del protocolo de transmisión usado.

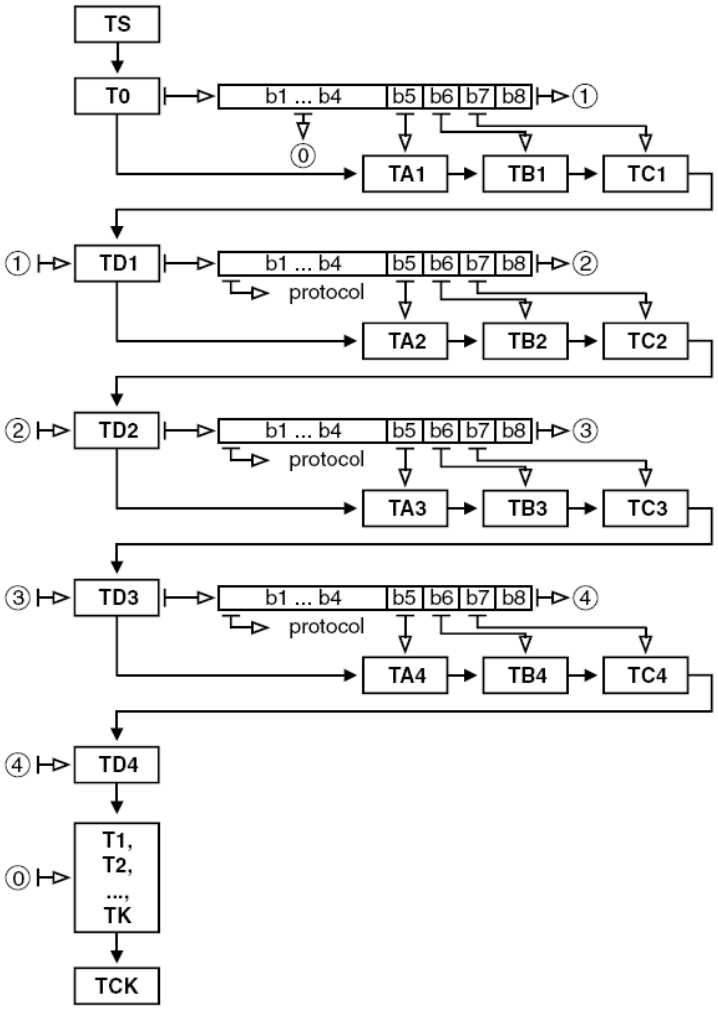


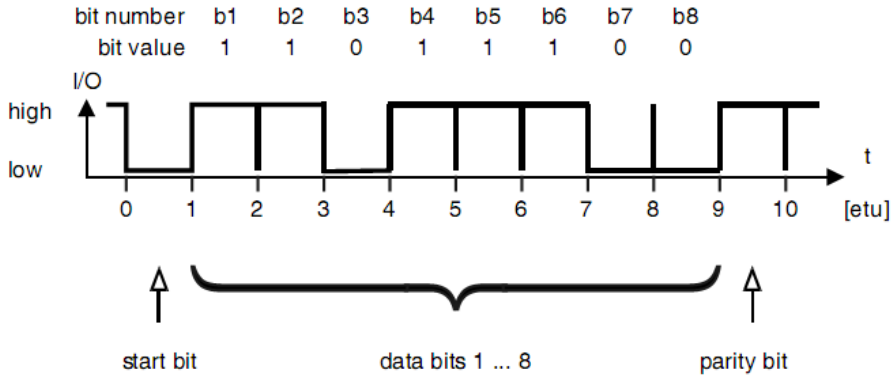
Table 6.1 The data elements of the ATR and their designations according to ISO/IEC 7816-3

Data element	Designation
TS	Initial character
T0	Format character
TA1, TB1, TC1, TD1, . . .	Interface characters
T1, T2, . . . , Tk	Historical characters
TCK	Check character

Caracteres del ATR:

El **carácter inicial, 'TS'**, especifica la convención que será usada para toda la transmisión del ATR y subsecuentes procesos de comunicación. Además, el byte TS contiene un patrón de bit característico que puede ser usado por el terminal para determinar el valor del divisor. Para este propósito, el terminal puede medir el tiempo entre dos flancos de bajada en el TS y dividir este tiempo entre tres. El resultado es la duración de un etu. Sin embargo, aunque el divisor esta fijado en 372 para el ATR, el terminal normalmente no evalúa el patrón de sincronización. El primer byte es una parte obligatoria del ATR y debe ser siempre mandado y solo dos códigos son permitidos para este byte,

‘3B’ para la convención directa y ‘3F’ para la inversa; en nuestro caso ha sido siempre la primera.
La siguiente figura muestra el diagrama de tiempos del carácter TS usando convención directa.



El **segundo carácter, T0**, contiene una serie de bits que indican cuales caracteres de interfaz siguen en el ATR. También indica la cantidad de caracteres históricos que siguen después de los caracteres de interfaz. Como TS, este byte es obligatorio y debe estar presente en todo ATR.
La tabla a continuación muestra el código del carácter de formato T0:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
…	…	…	…		X			Number of historical characters (0 to 15)
…	…	…	1	…	…	…	…	TA1 sent
…	…	1	…	…	…	…	…	TB1 sent
…	1	…	…	…	…	…	…	TC1 sent
1	…	…	…	…	…	…	…	TD1 sent

Los **caracteres de interfaz** especifican todos los parámetros de transmisión para el protocolo usado. Estos consisten en TAi, TBi, TCi y TDi bytes. Estos bytes son opcionales en el ATR y pueden ser omitidos en forma apropiada.
Estos caracteres se clasifican en caracteres de interfaz global y caracteres de interfaz específicos. Los primeros especifican parámetros básicos para el protocolo de transmisión, como lo es el divisor, que será usado para todos los protocolos subsiguientes. Los segundos especifican parámetros para un protocolo de transmisión particular. El “tiempo de espera de trabajo” para el protocolo T=0 es un ejemplo típico de dicho parámetro.

En principio, los caracteres de interfaz globales aplican para todos los protocolos, pero por razones históricas (desde que originariamente solo el protocolo T=0 era definido en los estándares ISO), muchos de estos caracteres son relevantes solamente para este protocolo. Si el protocolo T=0 no es usado, estos pueden ser omitidos, y en ese caso se aplican los valores preestablecidos.
Cada byte TDi es solo usado para enlazar a subsecuentes caracteres de interfaz. Para este propósito, el nibble superior de cada byte TDi contiene un patrón de bits indicando cual de los TA(i+1), TB(i+1), TC(i+1) y TD(i+1)

caracteres de interfaz siguen, usando el mismo código como para el carácter de formato T0. El nibble inferior de cada byte TDi identifica el protocolo de transmisión disponible en cada caso. Un byte TDi debe ser siempre enviado si cualquier subsecuente carácter de interfaz va a ser enviado.

Los demás caracteres (TAi, TBi y TCi), definen el /los protocolo/s de transmisión disponibles. El significado (de acuerdo con la norma ISO/IEC 7816-3) es el siguiente:

TA1:
El parámetro FI en el nibble superior codifica el divisor (factor de conversión de la tasa de reloj) F.
El parámetro DI en el nibble inferior codifica el factor de ajuste de la tasa de bit D.
Las siguientes tablas muestran estas codificaciones:

Table 6.5 Coding of TA1

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
		X						FI
		...						
					...			
					X			DI

Table 6.6 Coding of FI

F	372	372	558	744	1116	1488	1860	RFU
FI	0000	0001	0010	0011	0100	0101	0110	0111
f_{max}	4 MHz	5 MHz	6 MHz	8 MHz	12 MHz	16 MHz	20 MHz	...
F	RFU	512	768	1024	1536	2048	RFU	RFU
FI	1000	1001	1010	1011	1100	1101	1110	1111
f_{max}	...	5 MHz	7.5 MHz	10 MHz	15 MHz	20 MHz	...	...

Table 6.7 Coding of DI

D	RFU	1	2	4	8	16	32	RFU
DI	0000	0001	0010	0011	0100	0101	0110	0111
D	12	20	RFU	RFU	RFU	RFU	RFU	RFU
DI	1000	1001	1010	1011	1100	1101	1110	1111

Se aplican las siguientes relaciones:

- El intervalo de bit para el ATR y PPS, el que es llamado *etu inicial* esta especificado como:

1. $$initial\ etu = \frac{372}{f_i}\ \text{s}$$

- El intervalo de bit para el protocolo de transmisión usado después del ATR y PPS se puede definir en forma independiente del ATR. Este intervalo, que es llamado el *etu de trabajo*, es definido como:

1. $$work\ etu = \frac{1}{D}\frac{F}{f_s}\ \text{s}$$

El factor de ajuste de la tasa de bit (D) y el factor de conversión de la tasa de reloj (F) permite a la tasa de transmisión ser modificada y ser adaptada a circunstancias individuales. La frecuencia del reloj aplicada en las formulas se muestra como *f*.
El valor de frecuencia máximo permitido esta dado en el estándar como 5 MHz.

TB1:
Los bits b7 y b6 del TB1 codifican un factor de voltaje de programación, llamado 'II'.
Los bits b5-b1 definen el parámetro 'PI1'. El bit mas significativo, b8, siempre es 0 (no es usado).

Estos parámetros eran necesitados para la primera generación de tarjetas inteligentes, en la medida en que usaban EPROM para el almacenamiento de datos, en lugar de las EEPROM. La necesidad de altos voltajes y corrientes para la programación de EPROM debía ser proveída desde el terminal a través de contacto Vpp. Sin embargo, con el paso del tiempo, el avance en la tecnología permite que la codificación específica de de este byte pueda ser ignorada. Los parámetros PI1 y II tienen siempre el valor 0, lo cual indica que no se necesita voltaje de programación externo. Si el parámetro del TB1 es omitido en el ATR, el valor por defecto del Vpp de 5 V a 50 mA es aplicado, como es especificado en la norma.

TC1:

El TC1 codifica un tiempo extra de guarda, designado N, como un entero hexadecimal sin signo. Este tiempo de guarda extra es definido como una extensión de la duración del bit de parada. El valor de N indica cómo adicionales etu's deben ser adicionados al tiempo de guarda. TC1 es interpretado linealmente, excepto para cuando N = 'FF', el cual tiene un significado especial. Con el protocolo T=1, el normal valor de tiempo de guarda de 2 etus es cambiado a 1 etu si N='FF'. Con el protocolo T=0, el tiempo de guarda estándar de 2 etus es mantenido en este caso, para permitir que un error sea indicado por un nivel bajo sin el intervalo de tiempo de guarda.

La tabla a continuación muestra la codificación de TC1:

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
				X				Extra guard time N, with a range of 0–254 etu
1	1	1	1	1	1	1	1	X = 255 and T = 0: guard time = 2 etu X = 255 and T = 1: guard time = 1 etu

<u>Carácter de interfaz específico para el protocolo T=0:</u>

TC2:

TC2 es el parámetro final para el protocolo T=0. Este contiene el parámetro WI, el cual codifica el 'tiempo de espera de trabajo' ('work waiting time'). Este es el máximo intervalo entre los niveles altos de dos bytes consecutivos:

tiempo de espera de trabajo = (960*D*WI) etu de trabajo

Si el parámetro TC2 no esta presente en el ATR, el valor por defecto es WI = 10.

Los caracteres históricos:

Por mucho tiempo, los caracteres históricos no fueron definidos por ningún estándar. Como resultado entonces, estos contienen una extensa variedad de información, dependiendo del productor del sistema operativo.
Según la referencia [1], muchas compañías usan estos bytes para identificar el sistema operativo y la versión de la máscara de ROM (generalmente en ASCII). Estos caracteres no son requeridos para estar presentes en el ATR, de manera que pueden ser omitidos, sobretodo cuando se quiere que el ATR sea corto y rápido en enviarse.
La norma ISO/IEC 7816-4 provee para un archivo de ATR en adición a los caracteres históricos. Este archivo, con el identificador de archivo reservado '2F01', contiene datos adicionales para el ATR y están destinados a ser una extensión de los caracteres históricos, los cuales no pueden superar los 15 bytes.
Los parámetros en un ATR o en los caracteres históricos pueden contener información compleja relacionada con la tarjeta inteligente y el SO usado en ella; por ejemplo, pueden indicar cual selección de archivos o funciones de selección implícita son soportadas por la SIM y provee información sobre el mecanismo lógico del canal.
Pueden también tener información sobre el emisor de la tarjeta, o los números de serie de tarjeta o chip, etc. La codificación de objetos de datos relevantes esta definida en las normas ISO/IEC 7816-4 y ISO/IEC 7816-5.

El carácter de chequeo:

Este último byte en el ATR es el carácter de chequeo, el cual contiene la suma en forma excluyente (XOR) de los bytes, desde T0 hasta el byte anterior al carácter de chequeo. Esta suma de chequeo puede ser usada además del testeo de paridad para verificar la correcta transmisión del ATR. Sin embargo, a pesar de la aparente simplicidad de la estructura y computación de esta suma de chequeo, hay varias diferencias significantes entre varios protocolos de transmisión.
Si sólo es indicado el protocolo de transmisión T=0 en el ATR (como es nuestro caso) no es necesario enviar esta suma de chequeo al final del ATR. En este caso, simplemente no es enviada, ya que la detección del error en los bytes usando chequeo de paridad y retransmisión de los bytes erróneos es obligatorio en el protocolo T=0.

La siguiente tabla muestra un ejemplo de ATR para una SIM de tecnología GSM, con T=0 y convención directa (sacada de [1]):

Designation	Value	Meaning	Remark
TS	'3B'	direct convention	
T0	'89'	Y1 = '8' = °1000°	TD1 follows
		K = 9	9 historical characters
TD1	'40'	Y2 = '4' = °0100°	TC2 follows
		T = 0	T = 0 is used
TC2	'14'	WI = '14'	The work waiting time is '14'
T1 ... T9	'47' || '47' || '32' || '34' || '4D' || '35' || '32' || '38' || '30'		"GG24M5280"

Referencias:

[1]: Smart Card Handbook, Willey & Sons
Norma ISO/IEC 7816-3
Especificaciones Técnicas GSM 11.11 - 11.14
Documentación Terminal Para Tarjetas Inteligentes, Diseño Lógico II-2007