Cika Electrónica

Nota de Aplicación: CAN-026
Título: **BFS-2S Módulo de reconocimiento de huellas dactilares**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	07/02/05	

Les presentamos un novedoso módulo de reconocimiento de huellas dactilares, con capacidad de verificación 1:1 e identificación 1:N hasta N=8. También posee la posibilidad de subir y bajar información del módulo, por lo que es posible hacer verificación 1:N, siempre y cuando se identifique por otro medio, claro está.

Descripción del BFS-2S

El BFS-2S, de Aimgene, resuelve todo el problema de la captura, enrollment y verificación o identificación de las huellas dactilares. Se trata de un módulo en caja cerrada, con un par de LEDs bicolor, un sensor semiconductor de huellas dactilares, y un switch que se habilita al presionar con el dedo el área del sensor. La interfaz con el host es un port serie a 115200bps, a 3,3V, pero 5V-tolerant. El switch puede leerse en uno de los cinco cables que forman parte de la interfaz.

Los comandos y respuestas del BFS-2S son ASCII, legibles por el usuario. El comando es en sí un caracter imprimible, y la obtención del resultado de la operación se realiza mediante parsing de la respuesta, ubicando el área del resultado, en un formato fijo. Dado que la comunicación es a 115200bps, se deberá tener especial atención en disponer de un buffer importante y asignar correctamente las prioridades de interrupciones para que el port serie pueda recibir correctamente las respuestas. Es recomendable, además, esperar un cierto tiempo entre la respuesta de un comando y el envío de un nuevo comando.

El BFS-2S es un módulo al que tendremos que descubrir poco a poco, jugando con él. Lo mejor es aprovechar que sus comandos y respuestas son ASCII, conectarlo a un port serie de cualquier computador mediante un conversor de niveles TTL-232, y probar los comandos, observando la respuesta y familiarizándonos con las respuestas y modos de operación.

Descripción de la operación con huellas dactilares

Desde el host, es decir, el equipo que controla la operación, se envían los comandos que dicen al BFS-2 lo que tiene que hacer. Las operaciones principales son:

- captura
- enrollment en RAM
- enrollment en flash
- verificación contra RAM
- identificación contra flash.

Realiza además otra serie de operaciones, como subir y bajar la imagen de la huella dactilar, así como también del template, es decir, del producto de la operación de enrollment.

El proceso de verificación 1:1 consiste en validar que una huella dactilar que se toma se corresponde con la que se halla en memoria. Es el proceso más simple, y consta de dos partes, cada una de dos pasos:

Primera parte:
1. Captura de la huella dactilar
2. Enrollment en RAM

Segunda parte:

1. Captura de la huella dactilar
2. Verificación contra RAM

Por supuesto que cada vez que apaguemos el módulo, perderemos la información en RAM. Para solucionar esto, disponemos de dos comandos que nos permiten obtener la huella dactilar analizada (el template), para poder guardarlo en el host, y volver a ponerlo en el módulo, más adelante. Gracias a esto, es posible realizar verificación 1:N, si previamente se hace la identificación por otro método, es decir, de algún modo (obteniendo el nombre en una lista, usando RFID, palabras clave, etc) se "dice" de quien se trata, y mediante el módulo se verifica que sea quien dice ser.

El proceso de verificación 1:N, entonces, es similar al 1:1, sólo que hay un paso previo de construcción de la base de datos en la primera parte, y una búsqueda en dicha base en la segunda parte:

Primera parte:
➔ Repetir para cada individuo
   1. Captura de la huella dactilar
   2. Enrollment en RAM
   3. Pedido del template y almacenamiento en base de datos

Segunda parte:
1. Identificación del individuo por algún otro método y carga del template en el módulo
2. Captura de la huella dactilar
3. Verificación contra RAM

El proceso de identificación 1:N es levemente diferente, y hace uso de la flash del módulo, con capacidad de almacenamiento de hasta 8 templates. Mediante este proceso, es posible identificar a cuál de los templates disponibles se asemeja más una huella dactilar que se toma. Consta también de dos partes, a saber:

Primera parte:
➔ Repetir para cada individuo
   1. Captura de la huella dactilar
   2. Selección del número de template
   3. Enrollment en flash

Segunda parte:
1. Captura de la huella dactilar
2. Identificación contra flash

El módulo devuelve el número de template que se corresponde con la huella dactilar capturada.

Comandos

Veremos los comandos principales, a modo de clarificación de la hoja de datos. Las respuestas, en la gran mayoría de los casos, devuelven un string que visto en pantalla sería algo así:

```
Start  mb : sb
Result mb : rb
```

donde *sb* indica el resultado de la operación: si es igual a cero, entonces la operación se realizó correctamente; y *rb* es un dato que sólo tiene sentido en algunas operaciones, como identificación 1:N. Por ejemplo:

```
Start  mb : 00
Result mb : 01
```

El BFS-2S envía la respuesta a un comando una vez que termina de ejecutarlo, por lo que la misma puede llegar casi inmediatamente (en términos humanos), para el caso de una captura, o demorar unos 12 segundos, para el caso de un enrollment en flash. No hay minún tipo de polling o comunicación intermedia, por lo que

entre el envío del comando y la obtención de la respuesta, el host no tiene comunicación en el módulo BFS-2S y no puede determinar el estado del mismo.

Captura

El comando para la operación de captura es 'C', y es el que realiza la lectura de la huella dactilar. El resultado será *sb=0* a menos que por algún motivo no pueda leer correctamente la huella.

Ej.:

```
Start  mb : 00
Result mb : 0E    (carece de sentido y puede ser cualquier valor)
```

Enrollment en RAM

El comando para la operación de enrollment en RAM es 'R', y realiza el análisis de la imagen de la huella dactilar corriente, colocando el resultado en el buffer de trabajo del módulo. Este buffer es utilizado además por la operación de identificación, por lo que no es posible realizar una verificación luego de una identificación, a menos que se obtenga nuevamente el template, ya sea por carga o captura. El resultado será *sb=0* a menos que por algún motivo no pueda analizar la huella.

Selección de template corriente

Se selecciona mediante los comandos: '1', '2', '3', '4', '5', '6', '7', '8', es decir, el número del template. No existe respuesta, más allá de un retorno de carro y line feed.

Enrollment en flash

El comando para la operación de enrollment en flash es 'E', y realiza el análisis de la imagen de la huella dactilar corriente, colocando el resultado en el área de template corriente. El resultado será *sb=0* a menos que por algún motivo no pueda analizar la huella. A continuación del resultado, se imprime el número de template actual:

```
Start  mb : 00
Result mb : rb
template
```

Ej.: El número de template corriente es el 1

```
Start  mb : 00
Result mb : rb
01
```

Verificación contra RAM

El comando para la operación de verificación contra RAM es 'V', y realiza la validación de la huella capturada contra el resultado de la última operación de enrollment en RAM. El resultado será *sb=0* si se trata de la misma huella.

Ej.: La huella capturada coincide con la guardada en RAM

```
Start  mb : 00
Result mb : 0E    (carece de sentido y puede ser cualquier valor)
```

Ej.: La huella capturada NO coincide con la guardada en RAM

```
Start  mb : 0F
Result mb : 0E    (carece de sentido y puede ser cualquier valor)
```

Identificación contra flash

El comando para la operación de identificación contra flash es 'I', y realiza un análisis de proximidad entre la huella capturada y cada uno de los patrones (templates) almacenados, devolviendo como resultado el número de template que resulta más "cercano", es decir, el que parece concordar. El resultado será *sb=0* si se pudo obtener una comparación favorable, y *rb* contendrá el número de template que concuerda.

Ej.: La huella capturada coincide con el template número 5

```
Start  mb : 00
Result mb : 05
```

Ej.: La huella capturada no coincide con ningún template

```
Start  mb : 0F    (puede ser cualquier valor distinto de cero)
Result mb : 00    (carece de sentido y puede ser cualquier valor)
```

Borrado de templates

Para borrar un template, podemos utilizar el comando 'X', que borra el template corriente, es decir, el que fuera seleccionado por la última operación de selección de template. También podemos utilizar el comando 'D', que borra todos los templates. Las respuestas son diferentes al standard, y pueden observarse en la hoja de datos.

Manejo de templates e imágenes

Disponemos además de otros comandos para subir y bajar imágenes y templates. En todos los casos, el comando es seguido de un stream binario en uno u otro sentido, cuya longitud depende de la operación. En los casos en que se envía información al módulo, se recomienda insertar una demora entre el comando y los datos, como puede observarse en los ejemplos.

Template Módulo -> host: 'T', devuelve 392 bytes correspondientes al template generado mediante el comando de enrollment en RAM

Template Host -> Módulo: 'M', seguido de 392 bytes que ocuparán la misma área de memoria que un template generado mediante el comando de enrollment en RAM, contra el cual funciona el comando de verificación contra RAM

Imagen Módulo -> Host: 'B', devuelve 30400 bytes correspondientes a una imagen de 200x152 pixels, en 256 niveles de gris (8bpp grayscale).

Estos comandos no dan respuesta, más allá de la correspondiente entrega de datos, si corresponde, y un retorno de carro y line feed.

Software de aplicación

A continuación damos algunos ejemplos de uso en Dynamic C, originalmente desarrollados para Rabbit. Consideramos que la claridad del código sirve para que el usuario lo comprenda y pueda portarlo al procesador de su agrado. En todos los casos, se dimensionó un buffer como para obtener la totalidad del paquete de respuesta.

Envío de comandos y parsing de la respuesta

Se envía el comando requerido, luego se espera que el módulo responda. La respuesta tiene una longitud fija, pero por comodidad utilizamos una cofunción que vuelve luego de transcurrido un tiempo sin recibir datos, el cual seteamos lo suficientemente largo como para que no transcurra en medio del mensaje. Una vez obtenida la respuesta, como la misma será un string del formato mencionado:

```
Start  mb : sb
Result mb : rb
```

lo que tendremos en el buffer de recepción al finalizar un comando será:

```
0D 0A 53 74 61 7D 74 20 20 6D 62 20 3A 20 sb_hi sb_lo
0D 0A 52 65 73 75 6C 74 20 6D 62 20 3A 20 rb_hi rb_lo
```

donde *sb_hi* y *sb_lo* corresponden a los nibbles alto y bajo, respectivamente, del byte *sb*, que nos interesa.

Para el caso de un enrollment en flash, la respuesta será

```
0D 0A 53 74 61 7D 74 20 20 6D 62 20 3A 20 sb_hi sb_lo
0D 0A 52 65 73 75 6C 74 20 6D 62 20 3A 20 rb_hi rb_lo
0D 0A t_hi t_lo 0D 0A
```

Si analizamos el string que obtenemos en el buffer de recepción al finalizar un comando, tendremos entonces el resultado de la operación (valor de *sb*) en los caracteres 14 y 15; y el dato asociado (valor de *rb*, si corresponde) en los caracteres 30 y 31. En ambos casos, debemos convertir dos caracteres ASCII a un byte, que luego usaremos para chequear el resultado

```
char buffer[80];
int result,data;

cofunc void aimgene(int c)
{
int n;
   serDputc(c);        // envía comando
   wfd n=cof_serDread(buffer,sizeof(buffer),300);    // espera respuesta

   buffer[14]-=0x30;   // conversión de ASCII a binario
   if(buffer[14]>=10)
       buffer[14]-=7;
   buffer[15]-=0x30;
   if(buffer[15]>=10)
       buffer[15]-=7;
   buffer[30]-=0x30;
   if(buffer[30]>=10)
       buffer[30]-=7;
   buffer[31]-=0x30;
   if(buffer[31]>=10)
       buffer[31]-=7;
   result=(buffer[14]<<4) | buffer[15];      // obtención del resultado
   data=(buffer[30]<<4) | buffer[31];        // y dato asociado
//   printf("\nResult= %d, Data= %d",result,data);
//   buffer[n]=0;
//   puts(buffer);
}
```

Captura

Esperamos la presión sobre el switch (área del sensor) y luego enviamos el comando 'C'. El resultado (valor de *result*) deberá ser 0, a menos que algo haya impedido la correcta captura, como por ejemplo el retirar el dedo en medio del proceso.

```
cofunc void dedito()
{
       do {
              waitfor(BitRdPortI(PBDR,3));
              puts("\nListo para capturar tu dedito\n");
              waitfor(!BitRdPortI(PBDR,3));
              puts("Capturando...\n");
              wfd aimgene('C');
              if(result)
                     puts("\nPERO DEJALO PUESTO, APURADO !!!\n");
       } while(result);
}
```

Software ejemplo de verificación 1:1

Inicializamos la interfaz serie a 115200, 8 bits, sin paridad, 1 bit de stop. Enviamos el comando 'A', que solamente sirve para que el módulo responda con un string de identificación, luego esperamos que el usuario nos brinde su dedo para poder tomarle la huella dactilar, tarea que desempeña la función *dedito()*, que analizamos en el párrafo anterior. Luego esperamos un tiempo razonable, y le damos al módulo la orden de realizar el enrollment en RAM: 'R'. Una vez obtenida la respuesta, esperamos que el usuario (u otra persona) nos brinde nuevamente su dedo (u otro dedo), y procedemos a verificar si la nueva huella obtenida se corresponde con la aprendida, es decir, utilizamos el comando 'V'.

```
main()
{
	while(1){
		costate{
			serDopen(115200);
			wfd aimgene('A');
			wfd dedito();
			waitfor(DelayMs(1000));
			puts("OK, enrolling... (10 segundos)\n");
			wfd aimgene('R');
			puts("Listo, ahora verificamos\n");
			while(1){
				wfd dedito();
				waitfor(DelayMs(1000));
				puts("Capturado, verificando...\n");
				wfd aimgene('V');
			if(result)
				puts("No me gusta ese dedo\n");
			else
				puts("Parece ser el mismo\n");
			}
		}
	}
}
```

Software ejemplo de identificación 1:N

Tomaremos cuatro huellas dactilares, que para que la operación sea más interesante conviene que correspondan a diferentes dedos, si no a diferentes personas cada una. Primero realizamos la inicialización de igual forma que describimos para la operación anterior. A continuación, hacemos la selección del template corriente, enviando los números '1' a '4', según corresponda. Luego de seleccionar el template y esperar un tiempo razonable, pedimos al usuario que nos preste su dedo, y realizamos la captura de su huella dactilar, mediante el comando 'C'. A continuación, y luego de esperar nuevamente un tiempo razonable, procedemos a ordenar al módulo que haga un enrollment en flash, en el área correspondiente a este template, mediante el comando 'E'.

Una vez ingresadas las cuatro huellas, entramos en un loop en el cual esperamos obtener una huella dactilar, y mediante el comando 'I', le pedimos al módulo que nos diga de cuál se trata, dato que podemos obtener del valor de la variable *data*, que corresponde al número de template que más se asemeja a la huella con que comparamos.

```
main()
{
int i;
   while(1){
        costate{
		serDopen(115200);
		wfd aimgene('A');
		for(i=1;i<5;i++){
			waitfor(DelayMs(1000));
			wfd aimgene(0x30+i);
			printf("\nDame el dedo #%d\n",i);
			wfd dedito();
			puts("OK, enrolling... (10 segundos)\n");
			waitfor(DelayMs(1000));
			wfd aimgene('E');
		}
		puts("\nListo, ahora verificamos\n");
		while(1){
			wfd dedito();
			waitfor(DelayMs(1000));
			puts("Capturado, verificando...\n");
			wfd aimgene('I');
			if(result)
				puts("No me gusta para nada\n");
			else
				printf("Parece ser el dedo #%d\n",data);
		}
```

```
        }
    }
}
```

Manejo de templates

Veremos ahora un par de funciones para subir y bajar los templates del módulo, lo que nos permitirá hacer verificación 1:N, que no es otra cosa que verificación 1:1 luego de haber hecho una identificación por otro medio. Como la longitud del template es de 392 bytes, la variable global *buffer* deberá tener una longitud mayor; el resto del código de la función de envío de comandos (*aimgene()*) es el mismo. Los templates los guardaremos en un área de *xmem*, y para ello definimos un puntero al área, y emplearemos las funciones *xmem2root()* y *root2xmem()* para moverlos. Tomamos un template del módulo con el comando 'T', y lo ponemos con el comando 'M', sin olvidar insertar una demora entre el comando y el envío del template.

```
#define TSIZE 392

char buffer[400];
long templates;

cofunc void gettemplate(int i)
{
        serDputc('T');
        wfd cof_serDread(buffer,TSIZE,300);
        root2xmem(templates+TSIZE*i,buffer,TSIZE);
}

cofunc void puttemplate(int i)
{
        serDputc('M');
        xmem2root(buffer,templates+TSIZE*i,TSIZE);
        waitfor(DelayMs(1000));
        wfd cof_serDwrite(buffer,TSIZE);
}
```

Software ejemplo de verificación 1:N

Primero que nada, pedimos un área de memoria para alojar los templates. Este programa es un demo para ejemplificar el funcionamiento, por lo que trabajaremos sobre RAM. En una aplicación real, convendría salvar los templates en flash luego de la captura.

Luego de inicializar la interfaz serie, procedemos a tomar *NTEMPLATES* huellas, que por conveniencia, simpleza, y minimización de aburrimiento hemos determinado fijar en tres. Luego de realizar la captura y enrollment en RAM de cada dedo, igual que como hiciéramos al verificar 1:1, pero antes de tomar la huella siguiente y sin olvidar esperar un tiempo razonable, procedemos a pedirle al módulo que nos pase el template mediante *gettemplate()*, y lo guardamos en algún lado, a modo de base de datos de templates. En este caso será un simple buffer en RAM, el usuario final seguramente encontrará formas más sofisticadas y útiles de proceder. Luego procedemos a despejar el buffer de recepción de cualquier caracter adicional (retorno de caro, por ejemplo) que pudiera haber venido a continuación de los 392 bytes del template y estamos listos para la captura siguiente.

Una vez capturados todos los templates, podemos proceder a verificar. Pidiendo disculpas de antemano por el poco grado de sofisticación, nuestro sistema de identificación será reducido a ingresar mediante el teclado el número de template (coincidente con el número de orden de dedo capturado) contra el cual comparar, el cual nos dará el índice de la localización del template dentro de la base de datos (buffer en RAM), el cual podremos pasar al módulo mediante *puttemplate()*, para luego proceder a la captura y verificación, tal como hiciéramos en la aplicación anterior 1:1. A modo de innovación, y para comprobar el funcionamiento de la verificación sin andar seleccionando el template a cada rato, le damos al usuario *NTRIES* chances de poner el dedo correcto.

```
#define NTEMPLATES 3
#define NTRIES 3

main()
{
int i,j;
```

```
char c;

        templates=xalloc(NTEMPLATES*TSIZE);
        while(1){
                costate{
                        serDopen(115200);
                        wfd aimgene('A');
                        for(i=0;i<NTEMPLATES;i++){
                                waitfor(DelayMs(1000));
                                printf("\nDame el dedo #%d\n",i+1);
                                wfd dedito();
                                puts("OK, enrolling... (10 segundos)\n");
                                waitfor(DelayMs(1000));
                                wfd aimgene('R');
                                puts("OK, uploading...\n");
                                waitfor(DelayMs(1000));
                                wfd gettemplate(i);
                                serDrdFlush();        // remove extra garbage from serial port
                        }
                        puts("Listo, ahora verificamos\n");
                        while(1){
                                j=0;
                                do {
                                        puts("Contra que dedo debo verificar ?\n");
                                        c=getchar();
                                        i=c-0x31;
                                } while(i>=NTEMPLATES);

                                printf("Downloading template #%d...\n",i+1);
                                wfd puttemplate(i);
                                waitfor(DelayMs(1000));
                                serDrdFlush();        // remove extra garbage from serial port
                                do {
                                        wfd dedito();
                                        waitfor(DelayMs(1000));
                                        puts("Capturado, verificando...\n");
                                        wfd aimgene('V');
                                        if(result) {
                                                puts("No me agrada en absoluto\n");
                                                j++;
                                        }
                                } while(result && (j<NTRIES));
                                if(result)
                                        puts("Cambiemos de dedo...\n");
                                else
                                        puts("Parece ser el mismo\n");
                        }
                }
        }
}
```

Claro está que los programas de ejemplo son sólo ejemplos, alguno de los pasos intermedios podría fallar y debería ser necesario repetir la tarea. En cada punto en particular, es posible cerciorarse del resultado de cada operación chequeando el estado de la variable global *result*.

En el archivo adjunto encontrarán además un par de programas adicionales, que implementan un demo utilizando un display Powertip 320x240 con touch screen, conectado al bus, que además tiene la opción de mostrar la huella dactilar en pantalla (reduciendo los 256 niveles de gris a un simple blanco y negro). Obtendrá mayor información de cómo utilizar este display y la touch screen en las CAN-016 y CAN-021, las cuales le indicarán además en qué otras notas de aplicación dispone de mayor información.

Si encuentra difícil el uso de costates y cofunctions, obtendrá mayor información en el tutorial CTU-001 y en notas de aplicación anteriores, así como también en toda la literatura de Rabbit.