|  |
| --- |
|  |
| LABORATORIO DE SISTEMAS DIGITALES |
| Realización de un escritor de mensajes de texto con un PIC 16F887 |

|  |
| --- |
| Javier Domingo Cansino  03/07/2012 |

# Manual de usuario de la aplicación

Esta aplicación, aunque no implementada entera, permite al usuario mandar un sms numérico a un teléfono. Para ello, y para intentar simular en parte el funcionamiento de un teléfono móvil, se debe encender, desbloquear, navegar por el menú y rellenar los campos.

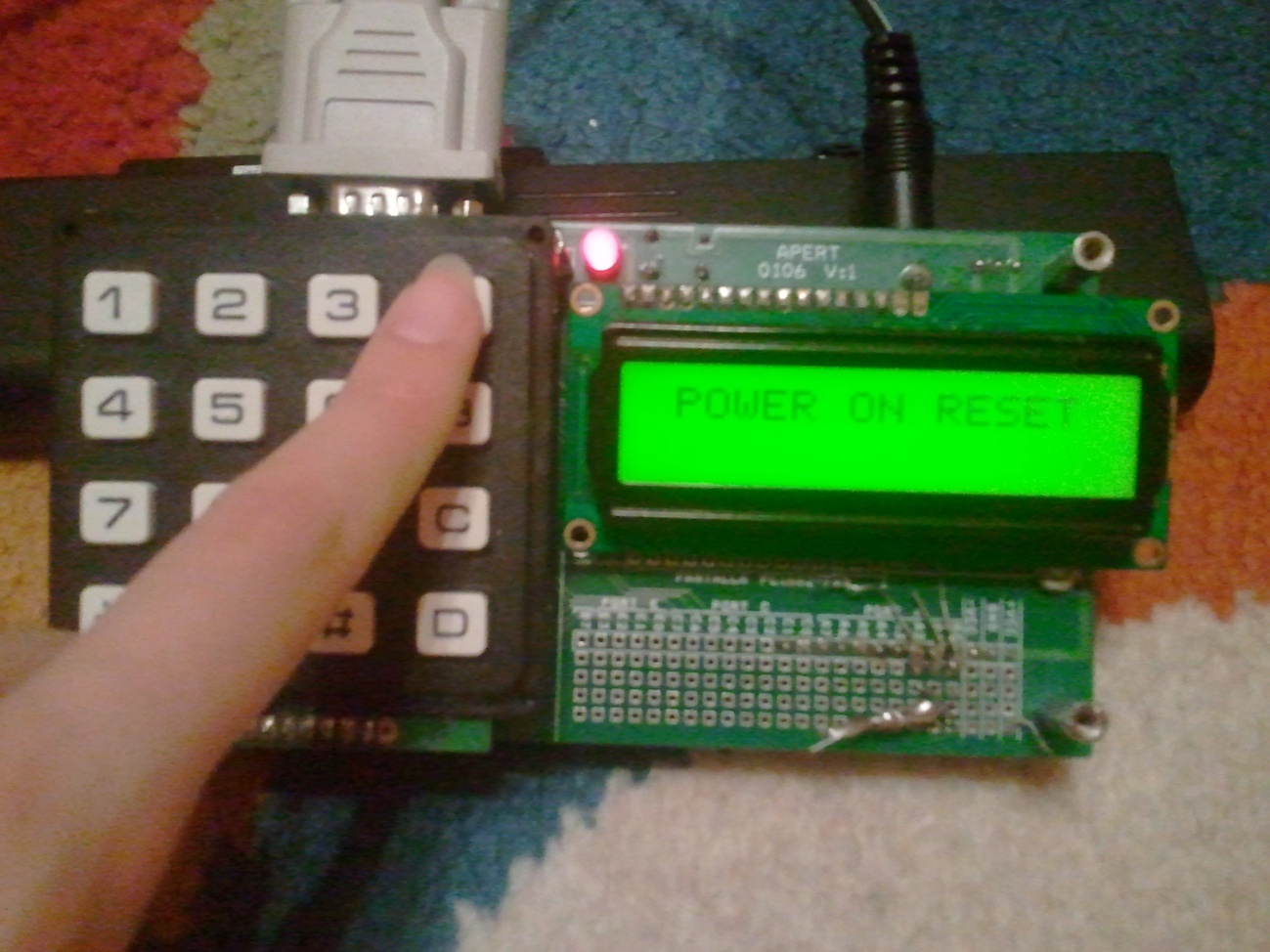
1. Encendemos el PIC, conectándolo al modem maestro 100 por puerto serie antes de que pasen 10 segundos.



1. Si hay algún error en la comunicación entre el PIC y el modem aparece *Power on reset* en la pantalla.



1. Por el contrario si no hay ningún error en la comunicación aparece *Standby*. Si esto no ocurre hay que dar a la tecla verde (letra A) para volver a intentarlo.





1. Al aparecer el *Standby* desbloqueamos el PIC. Para ello pulsamos las teclas \* y **#**. Primero pulsamos la **\***, sin soltarla pulsamos la **#**, después soltamos la **\*** sin soltar la **#** y por último soltamos la **#**.









1. Volvemos a pulsar la tecla verde esta vez para que nos aparezca el menú para escribir o leer mensajes.



1. Seleccionamos la opción de escribir mensaje pulsando la tecla verde.



1. Aparece en la pantalla *Marca el numero*. Por tanto se escribe el número al que se quiere mandar el mensaje y se da a la tecla verde.





1. Se procede entonces a redactar el mensaje que se desee enviar. Cuando el mensaje esté completo se vuelve a dar a la tecla verde para enviarlo.



1. En la pantalla aparece *Enviando* que cuando el mensaje se envíe cambiará a *Enviado!*





El Led rojo se enciende cuando el PIC detecta que hay una tecla pulsada.

# Memoria de la realización del trabajo

El trabajo se ha hecho utilizando un sistema de control de versiones, puediendo tener acceso a todas las versiones realizadas en cualquier momento. El trabajo para esta asignatura se puede encontrar en el repositorio Universidad, en <https://github.com/txomon/Universidad>, bajo la ruta 2/labSED. Ahí están todos los recursos utilizados. Tanto el código como los materiales adicionales, como diseños de placas para simulación etc.

Aunque la aplicación ahora mismo solo envía sms-es, se ha diseñado de manera que sea ampliable fácilmente. Por el momento, se han hecho módulos para trabajar con el puerto serie, la eeprom, la pantalla y el teclado.

A partir de estos módulos se ha construido en el resto de la lógica del programa hay que hacer especial hincapié en que lo más difícil o los más costoso de este trabajo no ha sido tanto la programación del programa de usuario que se ha ejecutado. Principalmente porque en este módulo había algunas máquinas de estados internas y externas que complicaban la programación. Se ha intentado hacer de la manera más compacta posible aprovechando al máximo las zonas de memoria que no se usaban de otros procesos.

La memoria como descripción del desarrollo lo he dividido en cinco partes principales. Las cuatro primeras explican cómo se ha llevado a cabo cada módulo teniendo en cuenta siempre que tienen que trabajar todos juntos y muchas veces de una manera asíncrona.

## La rutina del teclado

Las rutinas del teclado han sido hechas a mano utilizando interrupciones, el temporizador número dos y una leve modificación en la placa. Se ha tomado como referencia la rutina que se entregó en clase para su utilización.

En esta versión, se trabaja con interrupciones.

1. Se detecta que en el puerto asignado al teclado, el puerto B, ha habido un cambio de nivel de tensión. El puerto B esta diseñado para que salte una interrupción cada vez hay un cambio de tensión.
2. A través del vector de interrupción se llega hasta la rutina de atención al teclado. En esta rutina se programa en temporizador para crear una interrupción dentro de 40 ms.
3. A los cuarenta milisegundos se atiende a la rutina de servicio a la interrupción del temporizador número dos, en la que se comprueba tecla a tecla sí está pulsada o no. Gracias a los diodos que hay por el cambio de diseño en vez de las resistencias para evitar cortocircuitos, se ha evitado los cortocircuitos y además se ha posibilitado que se haga una lectura limpia de cada columna de teclas.
4. Hay dos registros que guardan cuáles son las teclas que están pulsadas en un cierto momento. Estas llevan los nombres KEYHL y KEYHU. Se encargan de llevar las 8 primeras teclas (las dos filas te abajo) y las 8 últimas teclas (las dos filas de arriba) respectivamente, poniendo a 1 los bits que correspondan a las teclas que están pulsadas y a 0 las que no.  
   De esta manera tan simple, se puede tener un teclado multitouch sin importar de qué teclas haya pulsadas.
5. El resto de aplicaciones solo tiene que leer esos registros para saber si las teclas que tienen están pulsadas o no.

## La rutina de la pantalla

Esta rutina está cogida de la que se reparte en clase. Se le ha modificado el código para estar particionado entre ficheros de cabecera y ficheros de programa. Además, se han añadido bastantes macros adicionales con la intención de tener todas las funcionalidades que permite la placa cubiertas a través de ellas. Construyendo a partir de las funcionalidades o macros básicas, funciones más complejas cuando es requerido.

También se han mantenido los nombres de las macros anteriores con intención de permitir a los profesores de la asignatura repartir el nuevo código con nuevas explicaciones en código, con intención de simplificar el entendimiento sobre el funcionamiento de la pantalla LCD.

Se ha creado una nueva función para uso personal que posibilita no tener que preocuparnos por la posición del siguiente carácter, sino simplemente enviarlo a la función, y ella se encarga de colocarlo adecuadamente. En caso de seguir con este desarrollo, lo importante sería la creación de una función que además de implementar esta funcionalidad con características avanzadas, como puede ser la rotación de líneas, también tuviera en consideración las palabras que no caben en una línea, etc.

## La rutina de la EEPROM

Estas rutinas iban ha ser más complejas, soportando una escritura encolada, simplifiqué su diseño por falta de necesidad, aunque se plantea un diseño dentro del fichero de cabecera.

Hay un registro que sirve para controlar si se va a utilizar una dirección externa o no, y otros dos para llevar el contador de las posiciones que se han escrito/leído.

Hay una función EEPROM\_WRITE y EEPROM\_READ que se encargan, dependiendo del registro de control, usar el contador de los registros, o utilizar en cambio la entrada que antes de la llamada a la función se ha proveído en la dirección asignada. Las llamadas a la EEPROM\_WRITE se encargan de asegurar al que las llama que pueda hacer llamadas a la función seguidas sin necesidad de tomarse el tiempo a esperar.

Esta perdida de tiempo la quería haber evitado haciéndolo a través de interrupciones y de una función que se encargara de crear un buffer en forma de anillo con los caracteres a escribir. De todos modos, a la hora de llevar a cabo las escrituras en EEPROM, he visto comprometida la integridad de la misma por no esperar los tiempos asignados.

## Las rutinas del puerto serie

El control del puerto serie se hace a través de interrupciones. La que más llama la atención es la rutina de envío, pero empezaré explicando la de recepción.

En memoria RAM, hay una memoria de 10 bytes dedicada a guardar los caracteres recibidos. Esta memoria, solo se suele leer cuando uno de los caracteres recibidos ha sido un CR, indicando en el protocolo AT fin de comando/respuesta normalmente.

La rutina de recepción por el puerto serie se limita a llevar un contador de la posición por la que está escribiendo en la RAM. No está preparada para la recepción de más de 10 caracteres por línea serie. Para utilizar la recepción por lo tanto, basta con mirar si el flag de caracteres recibidos está activado, y leer las posiciones que interesen (normalmente solo se lee un 0 y un CR).

La rutina de envío por el puerto serie es bastante más compleja. En definitiva, es una manera de conseguir enviar caracteres por un puerto de distintas zonas de memoria (Flash, Ram y EEPROM) sin necesidad de tener que estar pendientes de los cambios.

Esta rutina, coge los caracteres de los lugares estipulados, y se encarga de ir pasando de un lugar a otro enviando todos los caracteres de ese lugar. Con intención de mandar solo los caracteres que son, se suele poner un carácter a 0 en la posición de después del último a enviar. Este diseño es el que más merecía, ya que había otras maneras mucho más complicadas que permitían ahorrarse la posición de memoria de ese 0, pero utilizaban muchos más ciclos.

La manera de utilizarla es la siguiente. Se copian los caracteres a las zonas de memoria estipuladas, y se termina con 0. Después, se activan los flags de los correspondientes orígenes de datos (IS\_EEP, IS\_CMD, IS\_DAT) y por último, se activa el flag IS\_SND que es el que permite el envío. Como todas las transmisiones del puerto serie efectuadas comienzan por el comando AT, se ha creado una rutina SEND\_AT que se encarga de enviar el AT y de activar los flags necesarios para retransmitir de una manera seguida.

Por lo tanto, ahora que están las interrupciones para enviar por el puerto serie (que el registro de envío está vacío, y qué puede enviar), se utilizará la RSI para ir recorriendo uno a uno los flags y sus correspondientes zonas, poniendo a 0 el flag cuando se detecte el fin de una zona, permitiendo así pasar a la siguiente.

## Programa de usuario

Sobre los anteriores cuatro módulos, se compone el último módulo, que es el programa principal. Se compone de dos máquinas de estados, una que controla el estado, y otra el estado interno del estado en el que se está.

Hay siete estados, Power on reset, Standby, Unlocked, Menu[12]:1, Escribir numero, Escribir sms y enviar. Con nombres descriptivos y no uniformes, estos son los 7 estados de los que actualmente se compone el PIC. Las labores de cada uno están muy definidas, y normalmente suelen llevar unas pantallas características.

### Power on reset

Este es el estado por el que se entra a la máquina de estados. Se encarga de poner Power on reset, de enviar el comando *AT V0 E0*, que sirve para que el modem no haga echo, y de respuestas numéricas (para no saturar el buffer de recepción).

Después, en el otro estado interno, estará esperando la respuesta al comando, *0<CR>* que marca que el comando ha sido aceptado. Finalmente, dará paso al siguiente estado

### Standby

Este es el estado normal de los teléfonos móviles, y si todo ha ido bien, el usuario no llegará siquiera a ver el anterior estado, ya que él no requiere de ninguna entrada por parte del usuario, y solo se cerciona de la existencia del modem.

Standby por el contrario, se dedica a proteger a nuestro PIC de pulsaciones de teclas involuntarias, y aunque no existe aún el paso contrario (bloquear el teléfono), si que se ha implementado el típico desbloqueo de los teléfonos móviles.

Para desbloquear, se pasa por cuatro estados internos que empiezan por no haber nada pulsado, para luego solo estar pulsada la \*, después la \* y la #, y finalmente la # para finalizar soltándola. En este caso, si que se ha dotado a los estados internos de transición absoluta entre sí.

Al final, se pasa al siguiente estado, Unlocked.

### Unlocked

Estaba pensado para mostrar información sobre la calidad de la cobertura e información del operador, pero lamentablemente, no se ha podido llevar a cabo por limitaciones de los modem. En vez de suprimir esta pantalla, se ha preparado para su posterior implementación, y se ha dejado por un mensaje estático, y por su función de intermediario entre el menú y el estado bloqueado.

Para pasar al siguiente estado, dispone de dos estados internos que se limitan a comprobar que no hay nada pulsado, y después que el verde está pulsado. En este caso no se mira que carácter está siendo pulsado, sino tan solo la tecla.

### Menu[12]:1

Este menú estaba pensado para una de las anteriores funcionalidades que se le pensaba dar al PIC, pero que por sobredimensionamiento, se decidió suprimir dicha opción. Por lo tanto, como el anterior estado, tan solo se dedica a pasar al siguiente estado, aunque muestra otra opción que podría estar disponible.

### Escribir número

Aquí entra en juego un nuevo concepto de análisis de las teclas, se encarga de conseguir saber cual es el número (en forma de carácter ASCII) es el correspondiente a esa tecla. Se encarga de tener un estado 0 que solo se avanza cuando no hay ninguna tecla pulsada, y en el estado 1, solo se recorre entero cuando hay alguna tecla pulsada.

Entonces, se consigue su carácter asociado y se guarda en la RAM, en la memoria asignada para envío por serie, y también se muestra por pantalla. Así carácter a carácter. Cuando la tecla presionada es la verde, y la detecta como pulsada, entonces, se avanza al siguiente estado. También añade a la serie que se va a enviar los caracteres *"* y *<CR>* que dividen el número de teléfono con el mensaje en el comando.

### Escribir SMS

Una vez se ha decidido el número, se pasa a escribir el SMS. En esta versión, se utiliza el mismo parser para escribir el SMS. Dejando tan solo la posibilidad de mandar números como contenido del mensaje. Utiliza los mismos estados, con modificaciones para guardar los datos en la eeprom, y también añade al final un carácter *<SUB>*, insertado en el hyperterminal con *CTLR-Z* y que marca el final del cuerpo del mensaje.

### Enviar

Este es el último paso, y consiste en inicializar las variables correspondientes, como en activar los flags correspondientes a cada tipo de almacenaje.

Un mensaje de ejemplo quedaría entonces definido así:

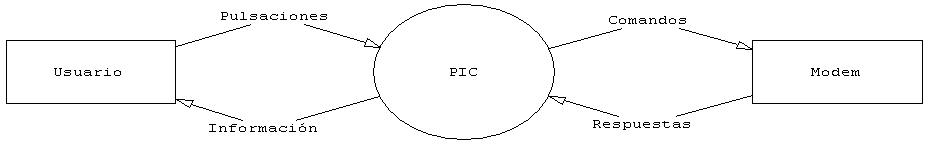
AT+CMGS="666111666"<CR>Este es un sms de ejemplo y en el programa estaría compuesto de dígitos numéricos <SUB>

Se encarga de enviarlo, y luego para analizar la respuesta, utiliza la misma función que se utilizaba al principio para analizar la respuesta del comando *AT V0 E0*. Hay una modificación hecha en la rutina de análisis para un caso muy especial de respuesta al comando de envío SMS.

Está preparado para en vez de buscar un 0 en la respuesta, buscar un *0<CR>*, ya que así evitamos problemas por la forma de la respuesta *+CMTI: 9<CR>0<CR>*.

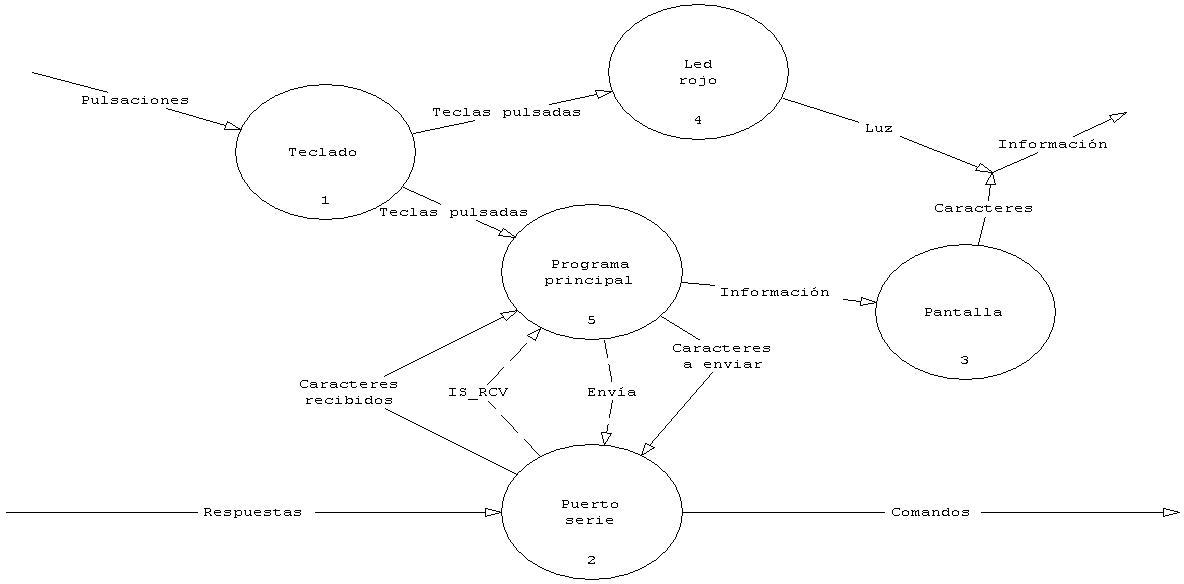
# Diagramas de flujo

## DFD0



Este es el diagrama de contexto de la aplicación. Se puede ver cuales son los elementos que interaccionan con el PIC.

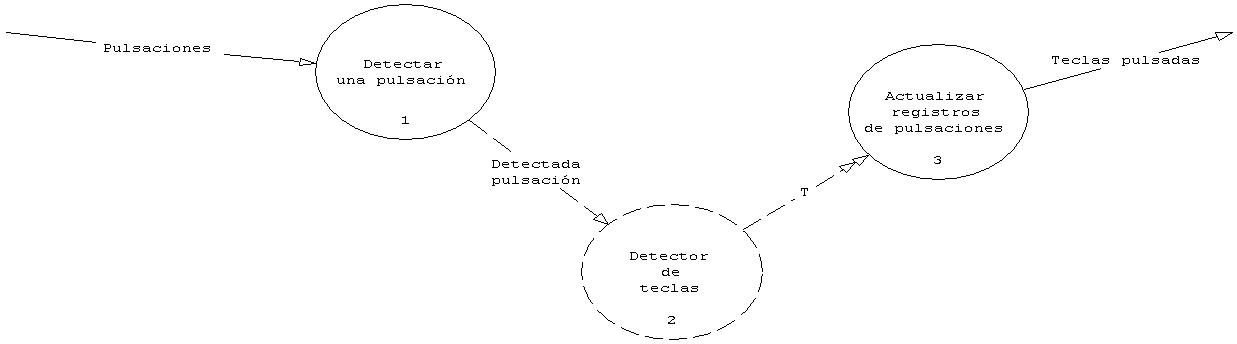
## DFD1



En este diagrama, que es el de primer nivel, se puede ver cuales son los módulos que hay dentro del PIC interaccionando con el programa principal. No se ha incluido el de la EEPROM porque no se utiliza el módulo descrito anteriormente siempre, ya que como se ha dicho, hubo problemas con él.

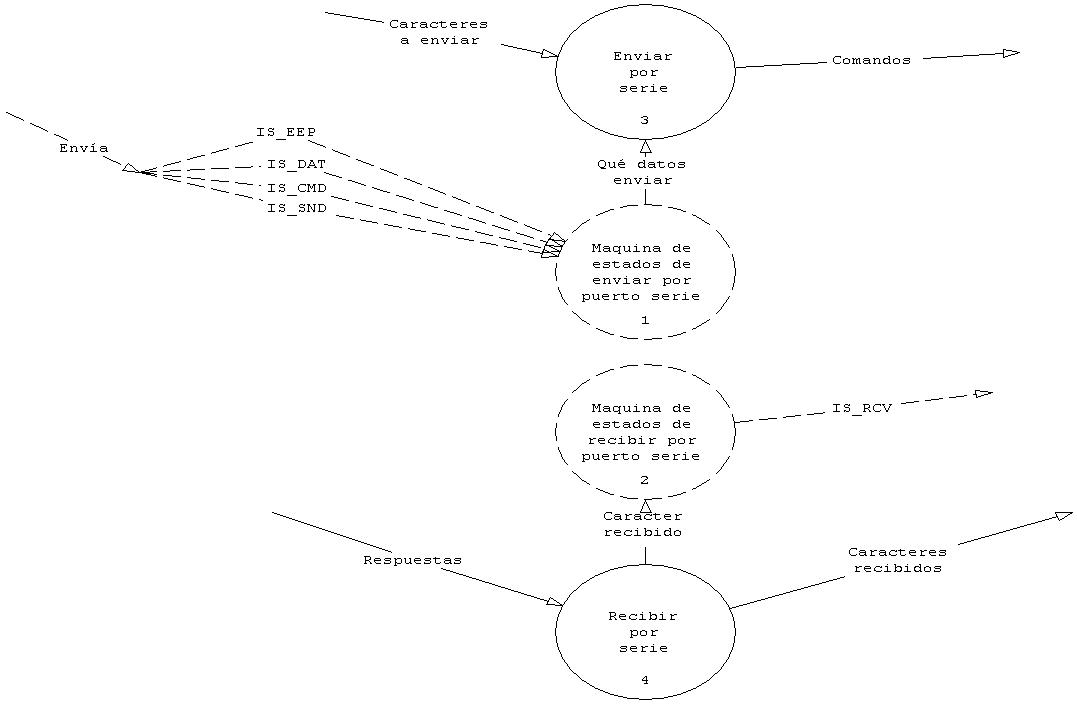
Se puede observar que la entrada y salida del usuario no son directamente hechas por el programa principal, y en cambio, se limita a dar instrucciones al resto de módulos para que funcionen

## DFD1P1



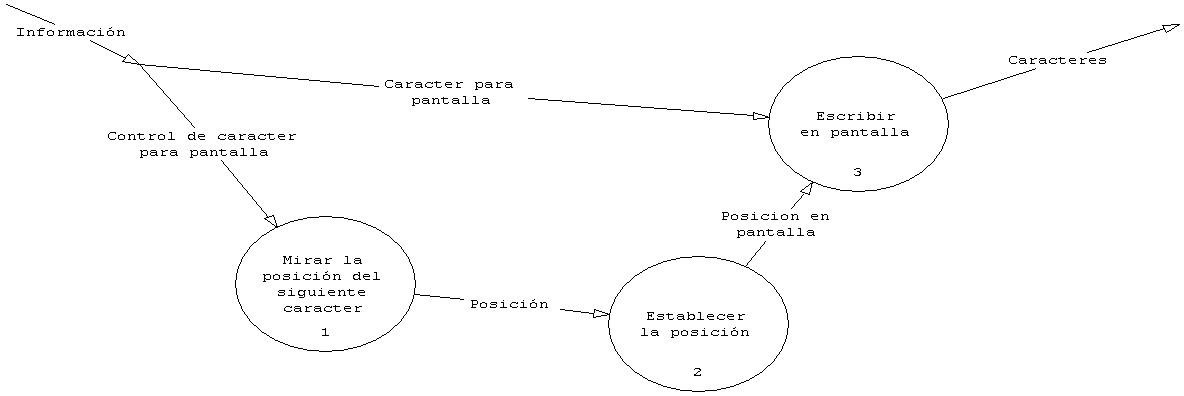
Este es el diagrama de flujo del proceso 1 del DFD1, el teclado. Se activa con pulsaciones, y cuando se detecta una pulsación, a través de las interrupciones disponibles, el usuario puede ejecutar el proceso de control Detector de teclas, que una vez haya decidido que teclas hay pulsadas, lanzará un trigger al actualizador de registros, que publicará los nuevos registros para que estén disponibles para el resto de procesos.

## DFD1P2



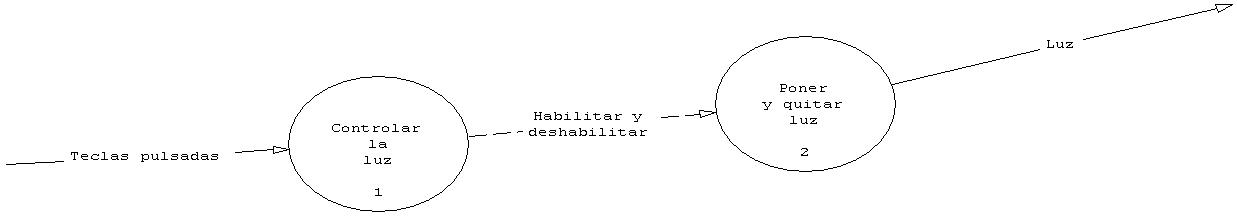
En este diagrama de flujo, podemos ver descrito el proceso 2 del nivel 1, el proceso que controla el puerto serie. Hay unas señales de control, tanto para el envío como para la recepción, y se encargan de organizar, a través de unas máquinas de estados, el envío o recepción de caracteres.

## DFD1P3



Este diagrama describe el proceso 4 del primer nivel. Es la descripción de la rutina de pantalla. Se ha suprimido la lógica específica de la implementación del PIC, y se ha dejado el flujo conceptual. He creado una función, que es la que suelo utilizar para escritura en pantalla, que me permite en algunos casos no tener que preocuparme por las posiciones de las escrituras cuidando no salirme fuera de la pantalla.

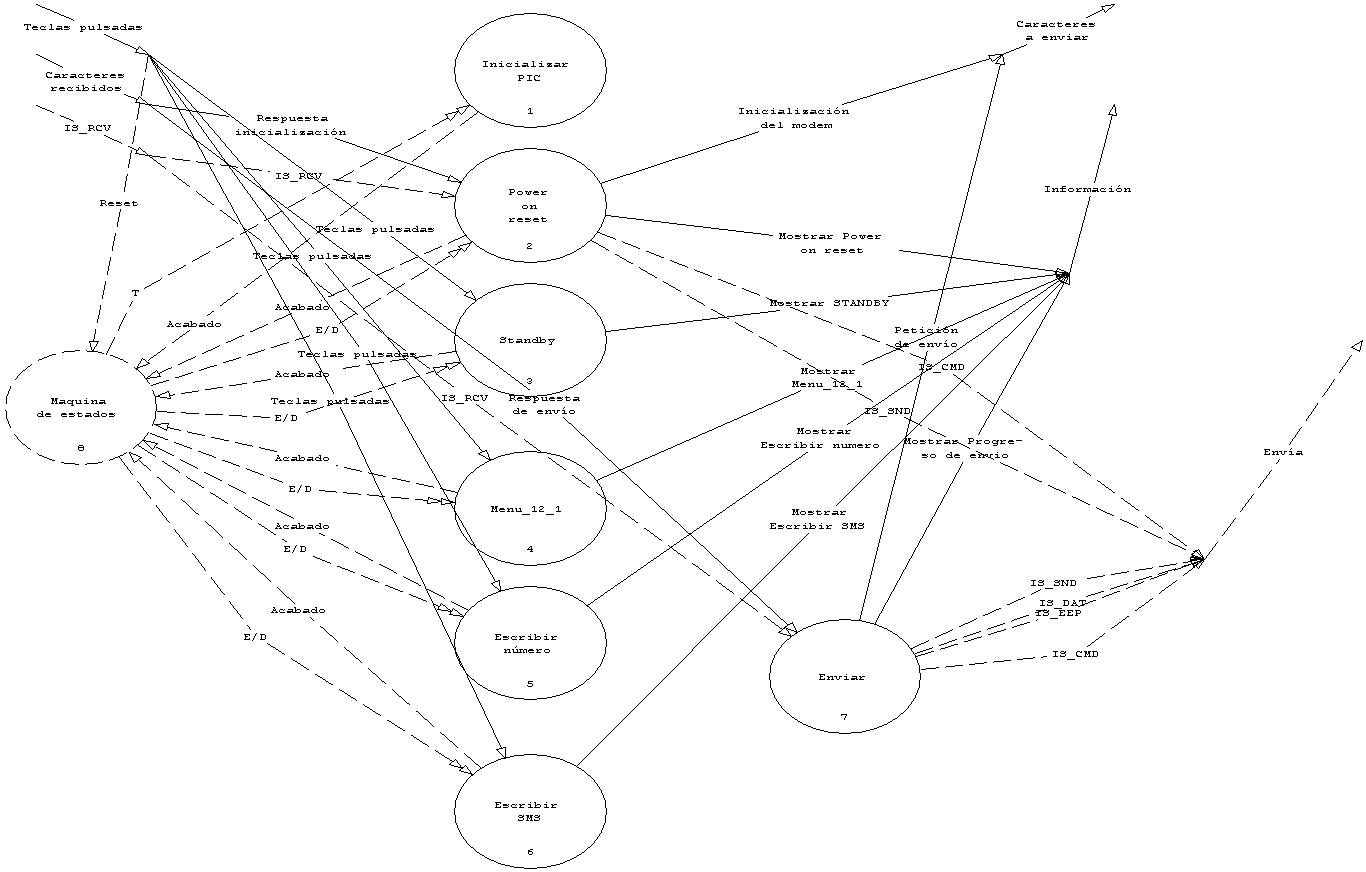
## DFD1P4



Este es el último diagrama de flujo de los módulos externos al programa principal. Se encarga de controlar el led rojo para dar información visual al usuario, y que sepa cuando está siendo admitida la pulsación de la tecla por el sistema.

Es una manera visual de proveer de información al usuario.

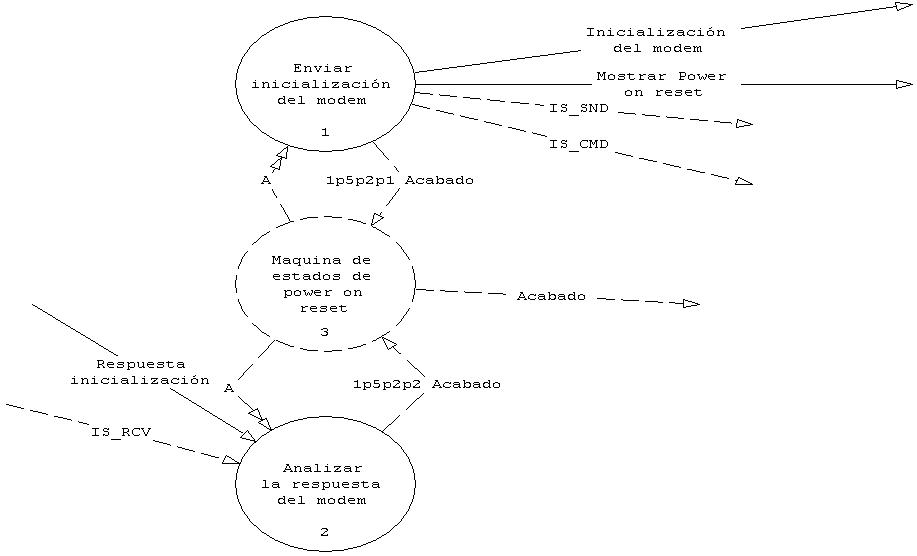
## DFD1P5



Este diagrama de flujo, es el resultado de la máquina de estados del programa, y las señales que le competen a cada uno de ellos.

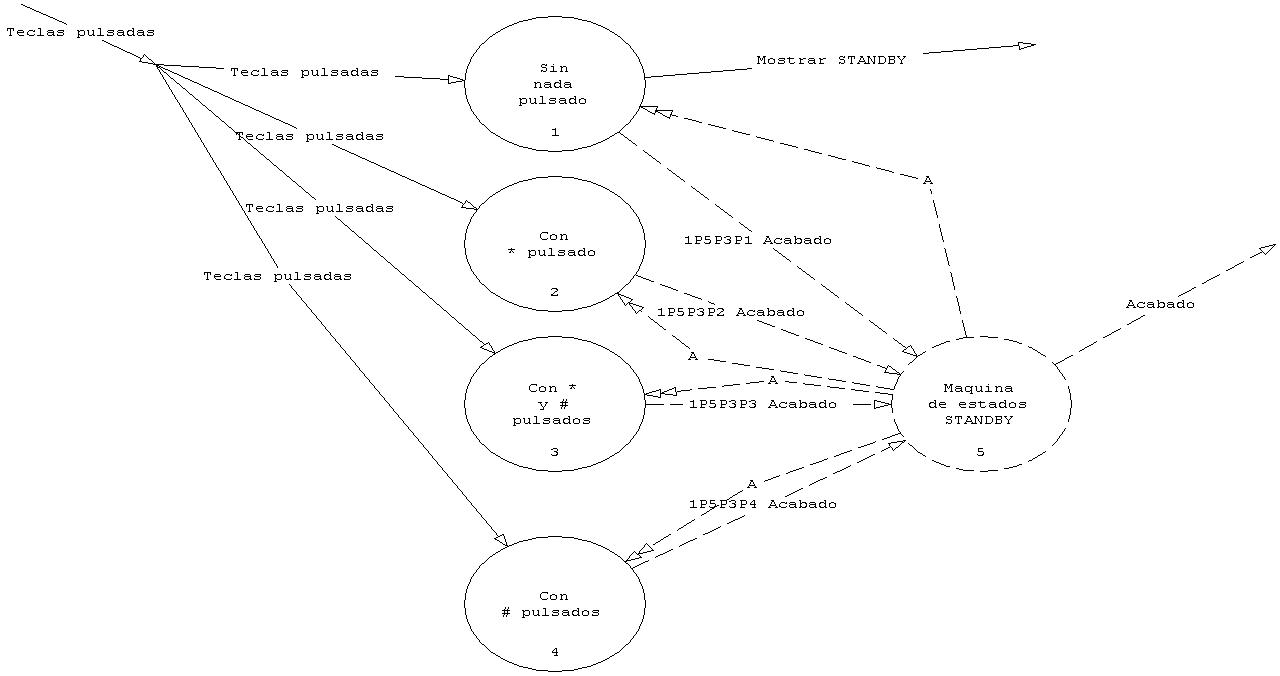
No se ha incluido un diagrama de flujo del programa del proceso número uno porque es un proceso que ejecuta todas las rutinas de inicialización de los elementos que va a utilizar en su ejecución el resto de procesos, y no tiene un flujo definido, sino una funcionalidad.

## DFD1P5P2



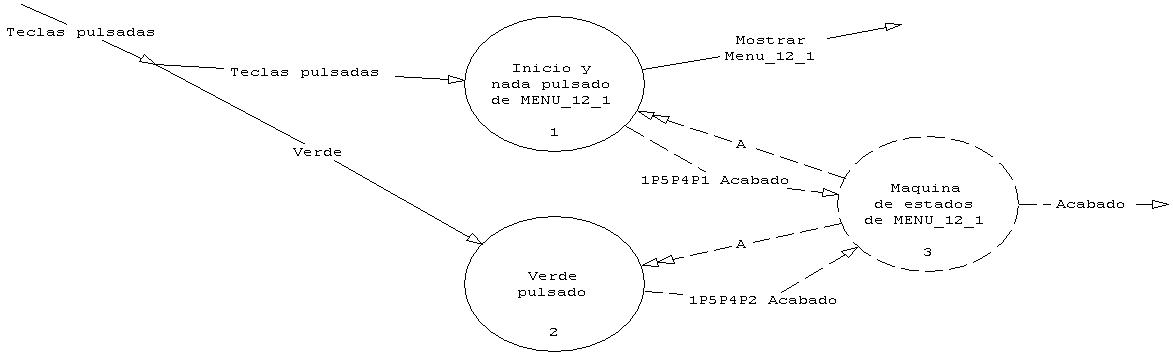
Esta es la descripción del proceso 2 del programa principal. Una vez activado, la máquina de estados interna se encarga de inicializar el proceso 1, y cuando ha acabado, activar el proceso de análisis de la respuesta, que dependiendo de la lógica y progreso del proceso, puede desembocar en mandar la señal Acabado a la máquina de estados general, para marcarle que puede pasarle el control a otro proceso, o volver a ejecutar el proceso 1.

## DFD1P5P3



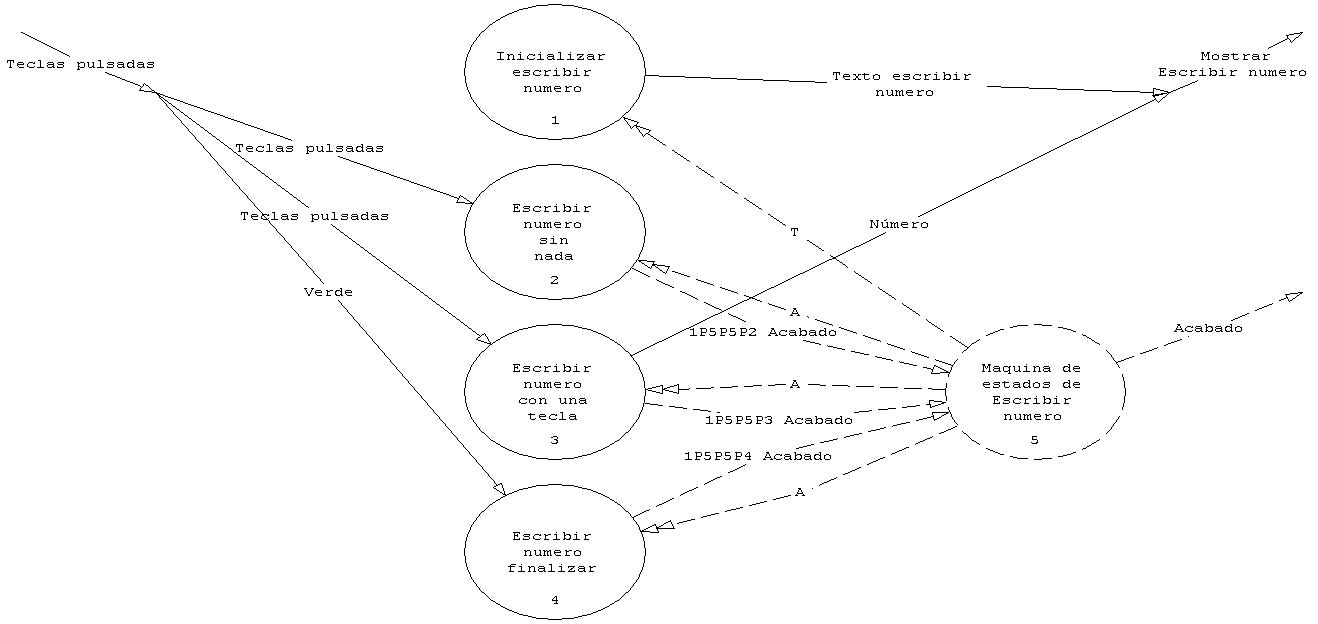
Este sería un ejemplo de la máquina de estados del teclado que se da para desbloquear el PIC. La máquina de estados va dando el control al estado que le compete, y cada uno hace lo que tiene que hacer, señalándole cuando han acabado.

## DFD1P5P4



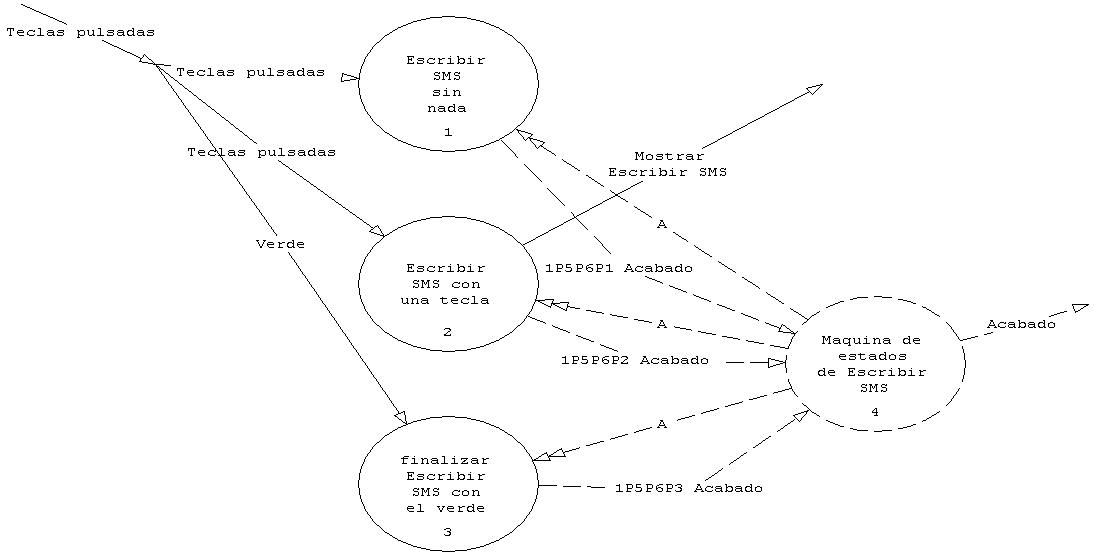
Este es un caso más simple que el anterior, en el que el primer estado tiene en cuenta todas las teclas, y el segundo solo la verde, pasando al siguiente estado nada más pulsado este.

## DFD1P5P5



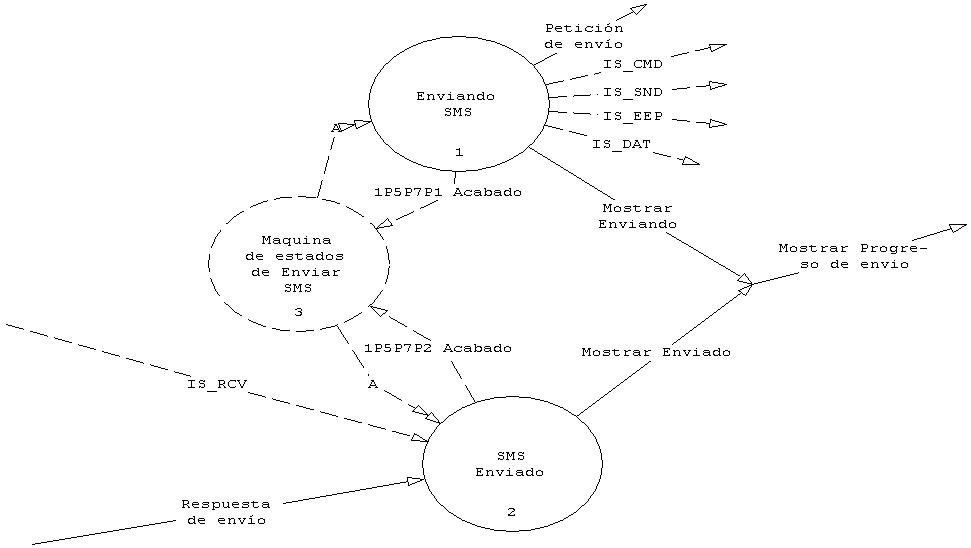
Aquí se explica el proceso de Escribir número , en el que hay 4 procesos, de los cuales solo se ejecutan 3 dentro de la máquina de estados, y el último es para inicializar diferentes datos de los que se tenga dependencia.

## DFD1P5P6



Es un diagrama muy parecido al anterior, pero no tiene fase de inicialización persé, si no que se apoya en lo hecho anteriormente para mostrar el mensaje correctamente.

## DFD1P5P7



Por último este es un diagrama de flujo bastante parecido al del proceso 2, solo que en este caso, no se envía tan solo una señal, si no que se envía la señal compuesta. El análisis se hace como en el primer caso (de hecho a la hora de implementar, así es).