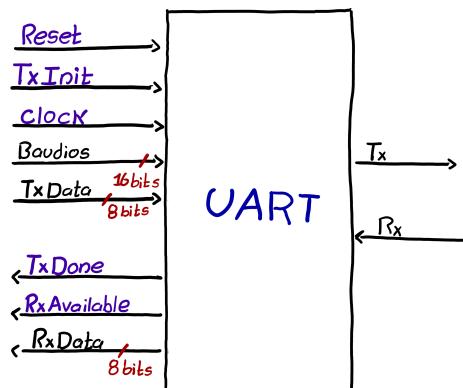
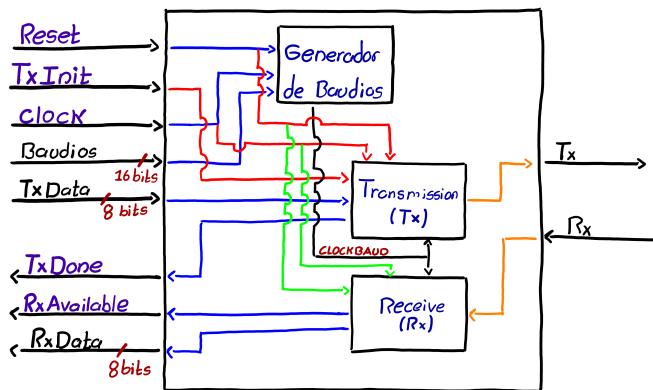


# Proyecto Electronica Digital — Uart

## Modulo Basico (Sin Bus)

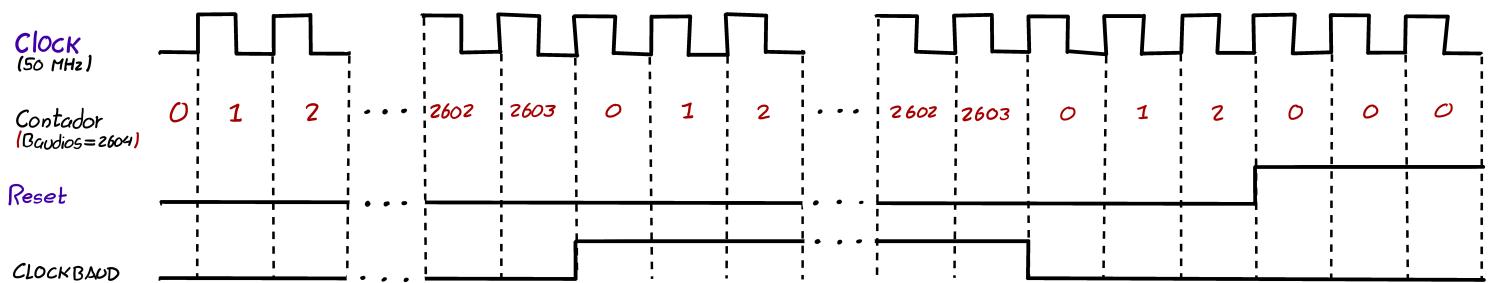


→ Diagrama Estructural



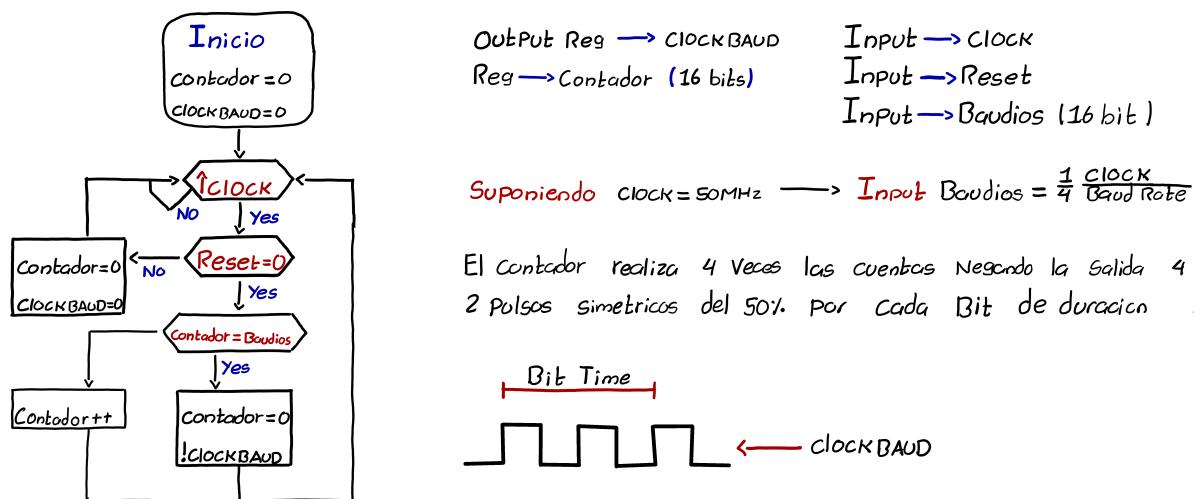
Generador de Baudios

→ Diagrama de Señales



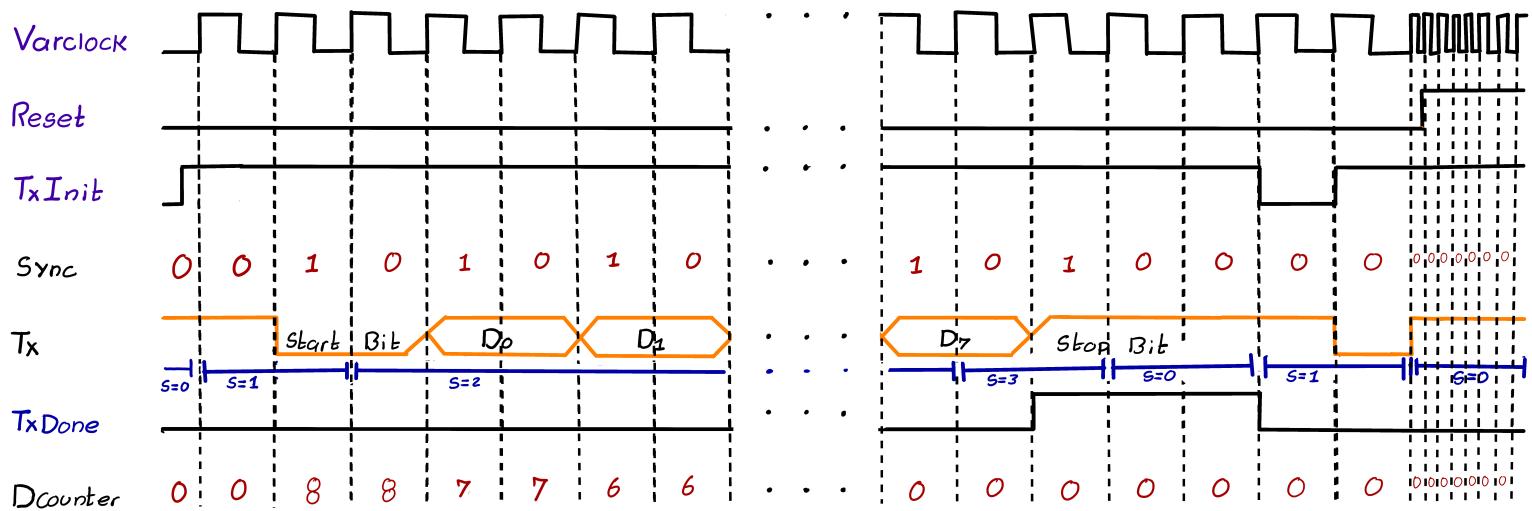
# Proyecto Electronica Digital — Uart

→ Diagrama Funcional



Transmission (Tx)

→ Diagrama de señales

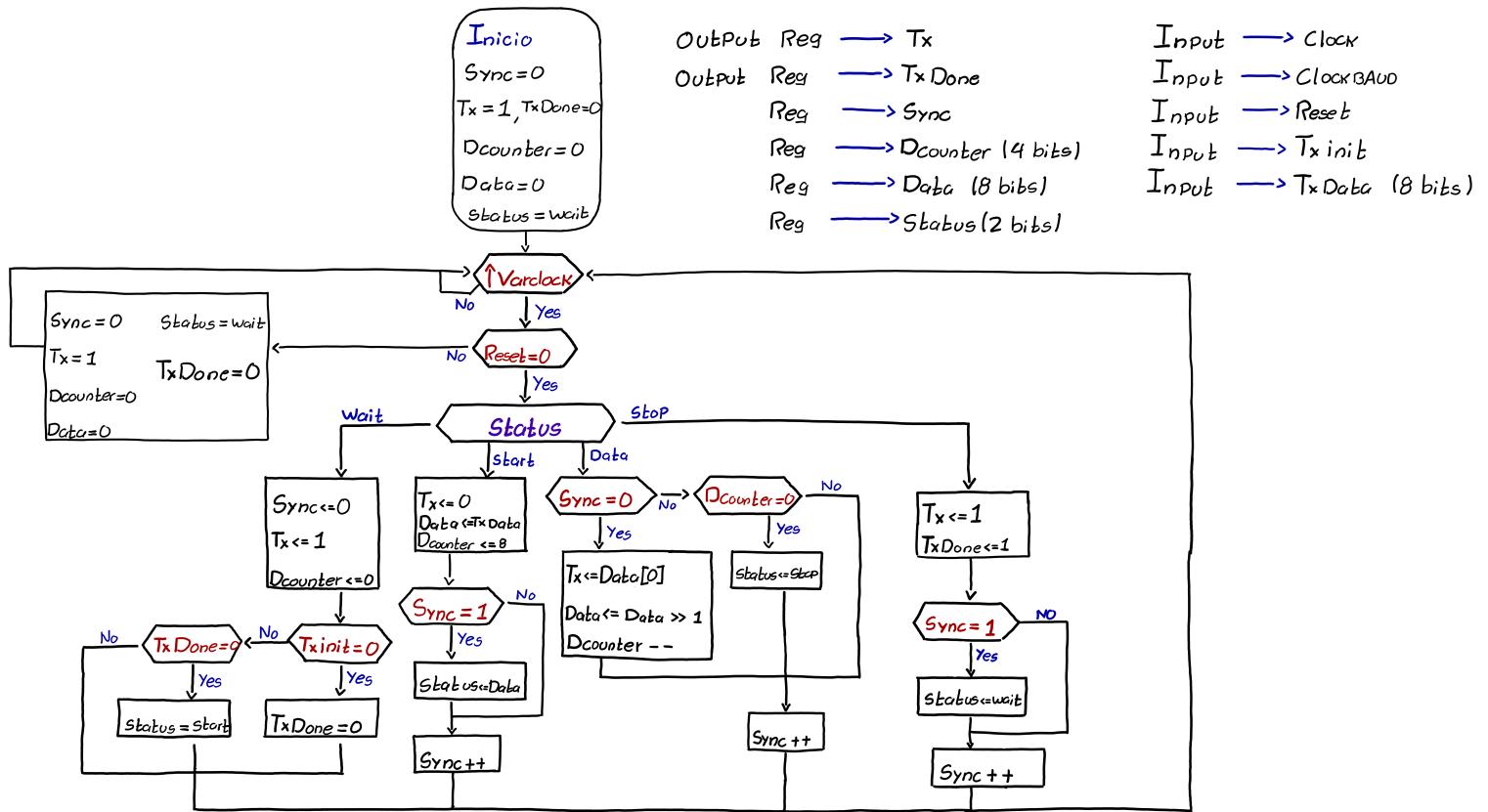


→ Estados:  $S=0 \rightarrow$  Wait  
 $S=1 \rightarrow$  Start  
 $S=2 \rightarrow$  Data  
 $S=3 \rightarrow$  End

Esta desincronización se debe al uso de la asignación no bloqueante. Por lo que los cambios se ven reflejados en el siguiente estado, de esta manera se puede garantizar que el bit de inicio dure su respectivo Bit-time y no la mitad de este.

# Proyecto Electronica Digital — Uart

## →Diagrama Funcional



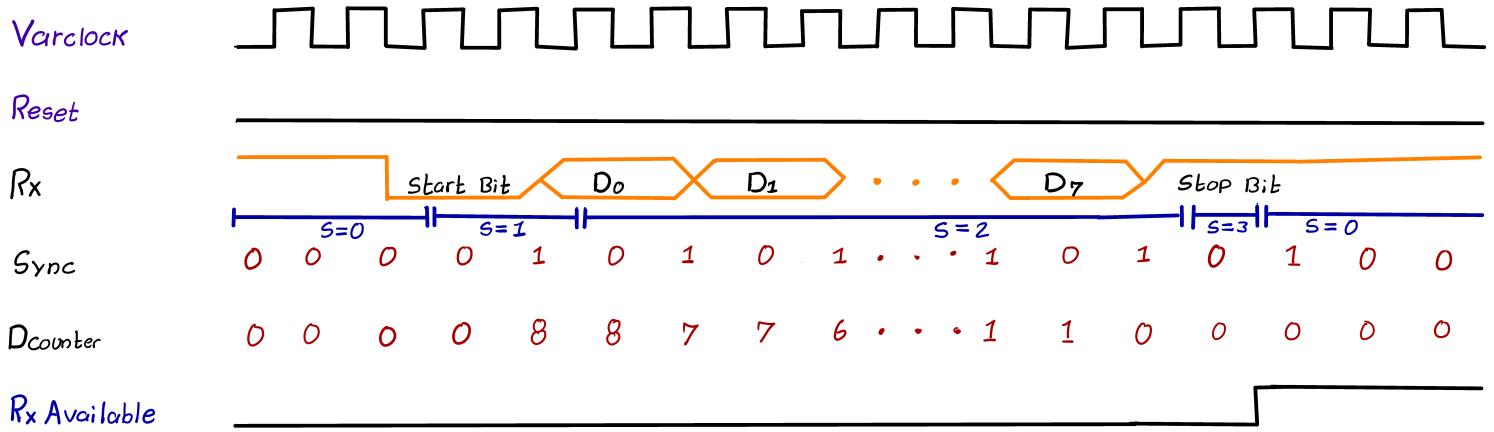
\* Debido a que el clock principal depende de un modulo que Divide frecuencia, este deje de funcionar con el reset. Por ende se planteo multiplexar los relojes para limpiar las Variables en situaciones de Reset usando el clock nativo de 50MHz

\* Para evitar los infinitos llamados se impuso la condicion de que para enviar un nuevo mensaje TxInib se debe deshabilitar cuando TxDone sea 1, de modo que cuando se vuelve 0 se podra volver a enviar un nuevo mensaje a traves de UART

# Proyecto Electronica Digital — Uart

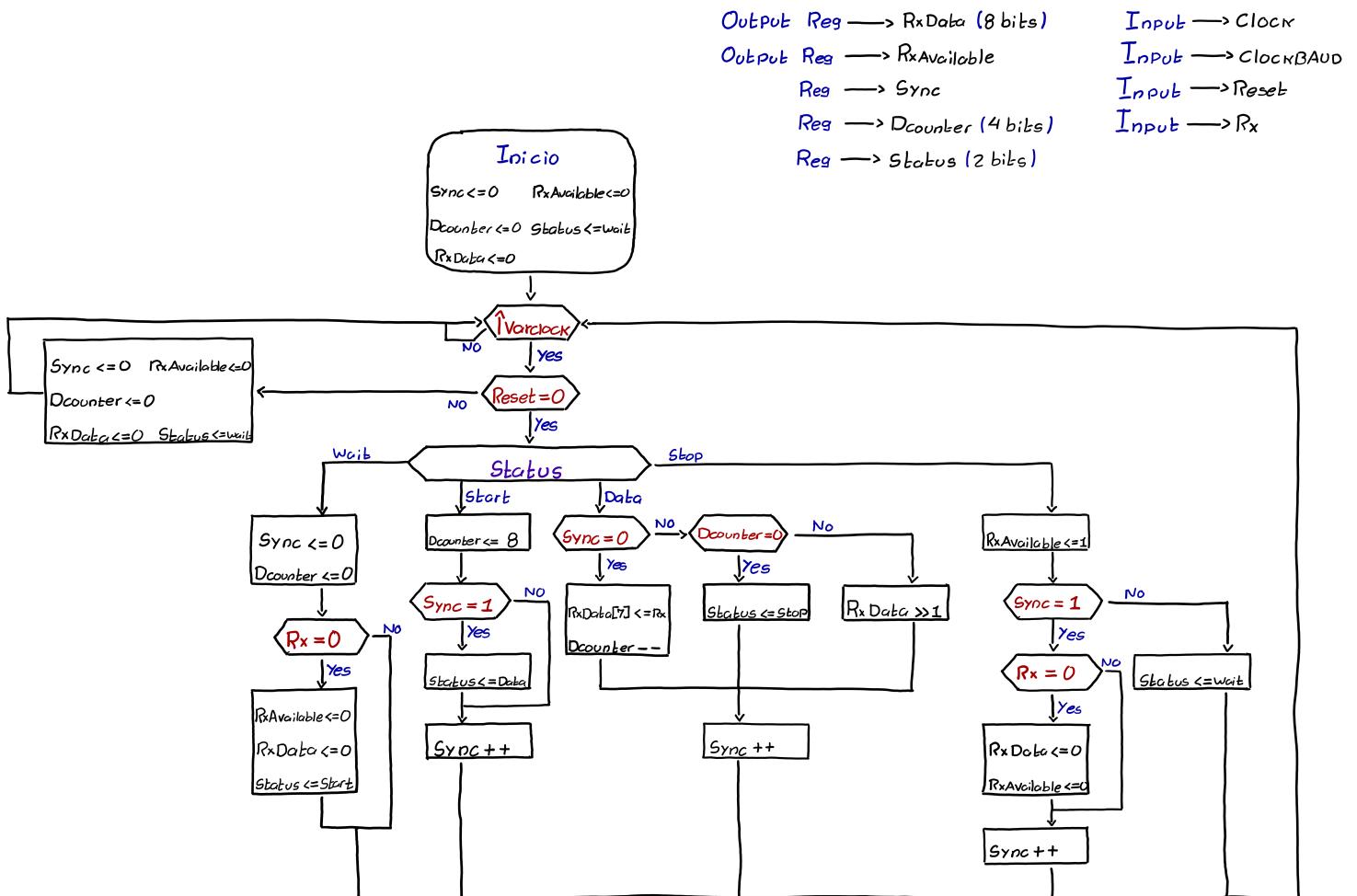
## Receive (Rx)

→ Diagrama de señales



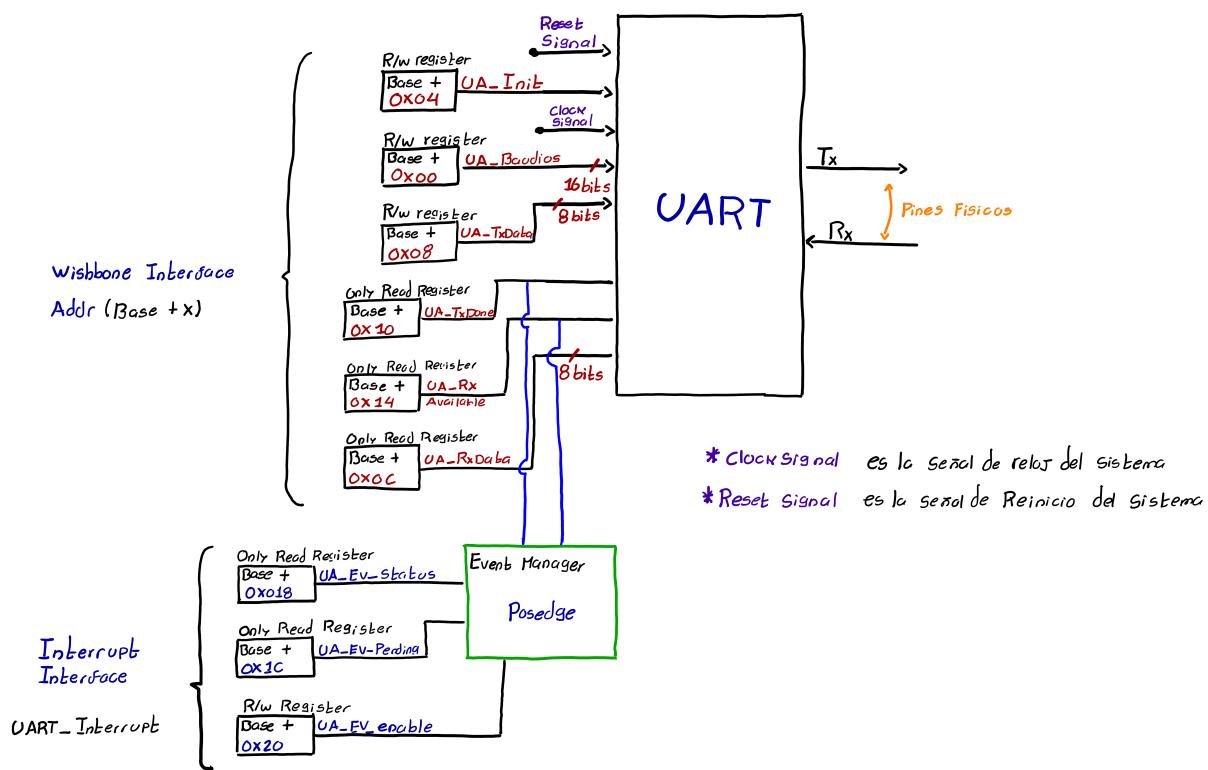
→ Estados:  $s=0 \rightarrow \text{wait}$   
 $s=1 \rightarrow \text{Start}$   
 $s=2 \rightarrow \text{Data}$   
 $s=3 \rightarrow \text{stop}$

→ Diagrama Funcional



# Proyecto Electronica Digital — Uart

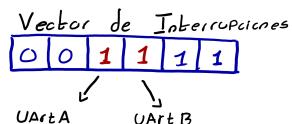
## Modulo wishbone (Incluye Interrupciones)



## Codigo C (Pic32 - Interrupciones)

- debido a que el modulo UART se implemento con el bluetooth y el df player mini se mostrara la funcion mas general y luego se haron las Variaciones para cada modulo.
- Uart tambien incluye interrupciones por ende se modifica **isr.c** agregando:

```
if(irqs & (1 << UARTA/B_INTERRUPT)) → Si la interrupcion proviene del Uart A o B
    UART A/B_Isr(); → ejecutar la funcion de Uart respectiva
```



# Proyecto Electronica Digital — Uart

→ Las funciones Generales Para hacer Posible el funcionamiento del UART

Static char RX\_Buffer[32]; → Arreglo que guardan los caracteres que se iran recibiendo

Static char TX\_Buffer[32]; → Arreglo que guarda los caracteres que se iran escribiendo

Static int TX\_Ocupado; → Contador que apunta al Caracter Actual que se escribe

Static int Tx\_Length; → Valor que almacena la longitud del mensaje a enviar

Static int RX\_Receive; → Apuntador del RX\_Buffer que Ayuda a saber cuantos nuevos caracteres hay por leer

Static int Baudios = 1302; → Valor de Baudios, este numero esta dado segun el baudrate y el clock del sistema

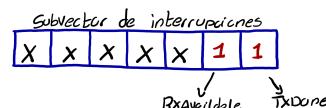
UARTAIB\_init() {

    UArbAIB\_ev\_Pending\_Write(UArt AIB\_ev\_Pending\_Read());

    UArbAIB\_ev\_enable\_Write(UARTAIB\_Ev\_TxDone || UARTAIB\_Ev\_RxAvailable); → habilita interrupciones del subvector de Interrupciones  
                        0x01 + 0x02 → 2'b11

    irq\_Setmask(irq\_Getmask() | (1 << UARTAIB\_Interrupt));

                        ↓  
                        habilita las interrupciones del  
                        UARTA o el UARTB en el  
                        vector general de interrupciones



    UARTAIB\_Baudios\_Write(Baudios); → Actualiza el registro (Base + 0x00) Para operar al valor de Baudios de la Variable

}

$$* \text{Baudios} = \frac{\text{Clock}}{4 \times \text{BaudRate}}$$

UARTAIB\_Baudios(int Baud){

    Baudios = Baud; → Actualiza el valor de Baudios en lo Rom

    UARTAIB\_Baudios\_Write(Baudios); → Actualiza el registro (Base + 0x00) Para operar al valor de baudios de la Variable

}

UARTAIB\_Set\_Baudios(){

    return Baudios; → Retorna el valor de la variable Baudios

}

UARTAIB\_Send(int length, char\* Data) {

    UARTAIB\_Baudios\_Write(Baudios); → Actualiza el valor del registro (Base + 0x00) Para Configurar la Velocidad de Transmision

    if(TX\_Ocupado == Tx\_Length) { → Condicion Solo Valida Cuando Ya Se termino de enviar un mensaje

        TX\_Ocupado = 0; → Apunta al Dato #0

        for(int i=0; i<length; i++) {  
            TX\_Buffer[i] = Data[i]; } → Carga el mensaje en el Buffer

    Tx\_Length = length; → Almacena en una Variable la longitud del mensaje

    UARTAIB\_TxData\_Write(TX\_Buffer[0]); → Actualiza el registro (Base + 0x08) Con el Primer Byte a enviar

    UARTAIB\_TxInit\_Write(1); → Inicia la Transmision Poniendo 1 en el registro (Base + 0x04)

}

    return;

}

UARTAIB\_Send\_Available(){

    if(TX\_Ocupado == Tx\_Length) { → Condicion Solo Valida si no hay un mensaje enviandose

        return 0; → Si el Protocolo de envio estan disponible retorna 0

    } else return 1; → Si no, retorna 1

}

# Proyecto Electronica Digital — Uart

UART A/B\_receive (int length) {

    if (Rx\_receive >= length) { → Si el mensaje del Buffer es mayor o igual a la longitud del mensaje que se requiere

        for (int k=0; k < (32 - Rx\_Receive); k++) Rx\_Buffer [Rx\_receive+k] = 0;

        Rx\_Receive = 0; → Regresa el Apuntador de la Posicion del Buffer a 0

    } } else return 0; → si no es mayor regresa 0



Borra el buffer de mensaje recibido  
Para las Posiciones mayores a la longitud

UART A/B\_receive\_All () {

    Rx\_Receive = 0; → Regresa el Apuntador de la Posicion del Buffer a 0

    return Rx\_Buffer; → Retorna todo el mensaje en el Buffer

}

Clear\_RX\_Buffer () {

    Rx\_Receive = 0; → Regresa el Apuntador de la Posicion de Rx\_Buffer a 0

    for (int i=0; i<32; i++) Rx\_Buffer [i] = 0; → Limpia todo Rx\_Buffer

}

UART A/B\_receive\_Available () {

    return Rx\_Receive; → Retorna el Apuntador (la longitud +1) del mensaje en el Buffer;

}

UART A/B\_isr () {

    int stat;

    stat = UARTA/B\_ev\_Pending\_read();

    if (stat & UARTA/B\_ev\_RxAvailable) { → Si la interrupcion es Porque llego un byte para leer

        Rx\_Buffer [Rx\_Receive] = UARTA/B\_RxData\_read(); → Guarda en el buffer en la posicion Rx\_Receive el dato del registro (Base + 0x0C)

        Rx\_Receive = Rx\_Receive + 1; → Apunta a la siguiente casilla del buffer

        if (Rx\_Receive == 32) Rx\_Receive = 0; → Si incrementa mas alla de la ultima posicion Vuelve al Principio

        UARTA/B\_ev\_Pending\_write(UARTA/B\_EV\_RxAvailable);

}

    if (stat & UARTA/B\_ev\_TxDone) { → Si la interrupcion es Porque termino de enviar un Byte

        UARTA/B\_ev\_Pending\_write(UARTA/B\_EV\_TxDone);

        UARTA/B\_Tx\_Init\_Write(0); → desactiva el envio de Datos Actualizando el registro (Base + 0x04)

        Tx\_Ocupado = Tx\_Ocupado + 1; → Apunta al Siguiente dato Para enviar

        while (UARTA/B\_TxDone\_Read() == 0) { → Lee el registro (Base + 0x10) hasta que Done sea 0, esto podria optimizarse

    } Con otra interrupcion Cuando Done cambie de 1 a 0

    if (Tx\_Ocupado != Tx\_Length) { → si el Apuntador del Buffer de envio es diferente a la longitud Aun faltan datos Para enviar

        UARTA/B\_TxData\_Write(Tx\_Buffer[Tx\_Ocupado]); → Escribe el nuevo dato a enviar Pues Tx\_Ocupado en este Puede

    es el siguiente Caracter Y lo Pone en el registro (Base + 0x08)

        UARTA/B\_TxInit\_Write(1); → Habilita el envio del nuevo dato Actualizando el registro (Base + 0x04)

    } else { → Ya se termino el envio del mensaje

        Tx\_Ocupado = 0; } inicializo en 0 las variables

        Tx\_Length = 0; }

}

}

# Proyecto Electronica Digital — Uart

→ Con esto en mente ya se tiene el Protocolo Para el envio y Recpcion de multiples bytes con tan solo enviar o leer una cadena de caracteres, estas funciones seran implementadas en los UART del modulo Bluetooth o el DF Player mini de manera que se combinen con funciones mas especificas para estos modulos.

## Modulo Bluetooth

Con la utilizacion de **UARTB\_Send** ya se pueden enviar mensajes si el dispositivo esta conectado y con **UART\_Receive** leer los mensajes en el Buffer de mensajes, Pero para configurar distintos Parametros del modulo se usan los Comandos **AT** Y por cada comando enviado se debe esperar una respuesta por lo cual los algoritmos implementados desaproven las interrupciones pero estos comandos suelen utilizarse solo en el inicio del programa.

→ Comando "**AT**": el comando **AT** se usa para saber si el bluetooth esta libre, Pues si no esta conectado a ningun dispositivo retorna "**OK**" y si esta vinculado responde "**OK+Lost**" desvinculando el dispositivo, Por ende se tiene el siguiente codigo.

```
blueooth_AT() {
    Char *Rx=0; → Variable que guardara el mensaje
    int delay=Baudios*2/25; → Tiempo de un Baudio  $\frac{10^6}{\text{BaudRate}}$  MS

    Clean_RX_Bufter(); → Limpia todos los mensajes de llegada
    While (blueooth_send_Available()==1){} → Esperar hasta que se pueda enviar un mensaje por uart
        UARTB

    blueooth_send(Strlen("AT"), "AT"); → Envia al modulo bluetooth el comando AT
        UARTB

    While (Rx==0){
        Rx=blueooth_receive(2); → hasta no recibir un OK (Ya sea solo OK o OK+Lost)
        UARTB

        delay_us(delay); → espera  $\frac{1}{\text{BaudRate}}$  S

    return Rx; → Retorna el OK
}
```

→ Comando "**AT+RESET**": El comando **AT+RESET** se usa para reiniciar el modulo bluetooth de tal modo que se hacen efectivos los cambios de la configuracion efectuada y se recibe el mensaje **OK+RESET**.

```
blueooth_Reset() {
    Char *Rx=0; → Variable que guarda el mensaje
    int delay=Baudios*2/25; → Tiempo de un baudio  $\frac{10^6}{\text{BaudRate}}$  MS

    Clean_RX_Bufter(); → Limpia todos los mensajes de llegada
    While (blueooth_send_Available()==1){} → Esperar hasta que se pueda enviar un mensaje por uart
        UARTB

    blueooth_send(Strlen("AT+Reset"), "AT+Reset"); → Envia al modulo bluetooth el comando AT+Reset
        UARTB

    While (Rx==0){
        Rx=blueooth_receive(8); → Esperar un mensaje de 8 bytes (OK+RESET)
        UARTB

        delay_us(delay); → espera un ciclo de BaudRate  $\frac{1}{\text{BaudRate}}$  S

    return;
}
```

## Proyecto Electronica Digital — Uart

→ Comando "AT+BAUDX": Configura la velocidad de conexión del UART entre el módulo y la FRESA dependiendo del valor de la tabla del módulo Bluetooth, luego responde OK+SET:X dependiendo de lo que se configuro.

```
bluetooth_Baud(int Baudmode){  
    Char *Rx=0; → Variable que guarda el mensaje  
    int delay=Baudios*2/25; → Retraso de  $\frac{10^6}{Baudrate}$   
    int baud=1302 → Variable que almacena el valor de baudios a cambiar  
    char Comando [8]; → mensaje del comando AT  
    char *BaudmodeCH=0; → Variable que almacena en char el modo de baudios a seleccionar
```

Clean\_RX\_Buffer(); → Se limpian los mensajes de llegada

```
switch (Baudmode) {  
    Case 0:  
        baud =  $\frac{50000000}{4 \times 9600}$ ; } Actualiza baud al valor correspondiente y Asigna el string a baudmodeCH respectivo  
        baudmodeCH="0";  
    break;  
    Case 1:  
        baud =  $\frac{50000000}{4 \times 19200}$ ;  
        baudmodeCH="1";  
    break;  
    :  
    Case 8:  
        baud =  $\frac{50000000}{4 \times 230400}$   
        baudmodeCH="8";  
    break;  
}
```

if(baud==Baudios) return; → Si el modo de configuración coincide con el actual sale de la función

strcat(Strcpy(Comando, "AT+BAUD"), Baudmode CH); → genera el string "AT+BAUDX" guardandolo en comando;

while (Bluetooth\_send\_Available ()==1) {} → Espera hasta que se pueda escribir un mensaje

UARTB  
Bluetooth\_Send(strlen(Comando), Comando); → Envía el comando "AT+BAUDX"

```
while (Rx==0){  
    Rx=Bluetooth_Receive(8); → Espera a recibir un mensaje de 8 bytes "OK+SET:X"  
}  
delay_us(delay); → introduce un retraso de  $\frac{10^6}{Baudrate} \times 4$   
Printf("%s\n", Rx); → Manda en consola  
return baud; → Retorna el valor de baudios  
}
```

## Proyecto Electronica Digital — Uart

→ Comando "AT+NAME X": El comando AT+NAMEX sirve para cambiar el nombre del dispositivo y el mensaje que el módulo retorna es OK+NAME:X donde el nombre máximo es de una longitud de 12 caracteres.

```
bluetooth_NAME (char * NAME) {
    char * Rx = 0; → Variable que guarda el mensaje
    char SendName [7 + strlen(NAME)]; → Variable que Almacena el comando completo

    StrCat (strcpy (SendName, "AT+NAME"), NAME); → Guarda el comando AT+NAMEX en la variable SendName
    int delay = Baudios * 2 / 25; → Tiempo de un bautio  $\frac{1}{BaudRate}$  S

    Clear_Rx_Buffer (); → Limpiamos todos los mensajes de llegada
    while (bluetooth_Send_Available () == 1) { } → Espera hasta que se pueda enviar un mensaje
        UARTB

    bluetooth_Send (strlen(SendName), SendName); → Envía el Comando "AT+NAMEX"
        UARTB

    while (Rx == 0) {
        Rx = bluetooth_Receive (7 + strlen(NAME)); → Espera respuesta "OK+NAME:X"
    }

    printf ("%s \n", Rx); → Muestra en consola el mensaje
    return;
}
```

```
bluetooth_Config_Baudmode (int baudmode) {
    char * Rx = 0; → Variable que guarda el mensaje
    int baud = 0; → Variable que guarda el valor de bautio a actualizar

    Rx = bluetooth_AT(); → Envía AT y Recibe OK/OK+LOST
    if (strcmp(Rx, "OK") != 0) { → Si el mensaje recibido no es OK
        printf ("%s \n", Rx); → Imprime que fue lo que se obtuvo
        return; → Sale de la función
    }
}
```

baud = bluetooth\_baud(Baudmode); → Guarda el valor de bautios obtenido enviando el mensaje AT+BAUDX y si el valor es diferente al modo actual y se recibe el mensaje OK+BAUDX

```
if (baud == 0) {
    printf ("Valor Actual"); → Muestra en consola que este es el Valor Actual
    return; → Sale de la función
}
```

bluetooth\_reset(); → Envía AT+RESET y Reiniciando el módulo hace efectivos los cambios recibiendo OK+RESET

```
UARTB_Baudios_Write (baud); → Actualiza el valor del registro (Base + 0x00) Para cambiar los bautios
Baudios = baud; → Actualiza la Variable Global de bautios
}
```

Proyecto Electronica Digital — Uart

```

blueooth_Config_Name (char *Name) {
    Char *Rx=0; → Variable que Guarda el mensaje

    Rx = blueooth_AT(); → Envia AT y Recibe OK/OK+Lost
    If(strCmp (Rx, "OK") != 0) { → Si no se Recibe OK
        PrintF ("%s \n", Rx); → Imprime el mensaje obtenido
        Return; → Sale de la funcion
    }

    blueooth_name (Name); → Envia el mensaje AT+NAMEX y Recibe OK+NAMEX de manera que Configura el nombre del modulo
                            blueooth

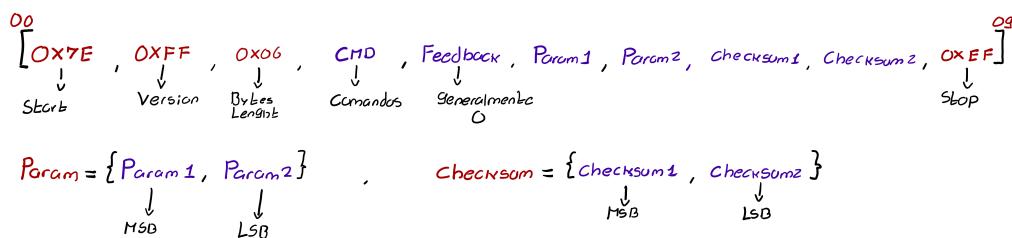
    blueooth_Reset (); → Envia el mensaje AT+RESET y Recibe OK+RESET, de manera que Reinicia el modulo y hace efectivos los cambios
}

```

## Modulo de Player Mini

En este caso se utiliza principalmente la función **UARTA\_Send** Para enviar comandos en el df Player mini. Ademas de que se redujo la cantidad de bytes de los buffers de envío y recepción A 10 Pues sera la longitud maxima de comandos que se utilizaran

Ademas el Comando C enviar tiene la siguiente estructura:



El valor de Parámetro se utiliza en aquellos comandos que requieren valores concretos, el checksum es un hash de seguridad. Por lo cual se debe calcular por cada comando su calculo es el siguiente:

$$\text{Checksum} = \text{0xFFFF} + \text{0x01} - (\underbrace{\text{Version}}_{\text{0x1F}} - \underbrace{\text{Length}}_{\text{0x06}} - \text{CMD} - \text{Feedback} - \text{Param1} - \text{Param2})$$

Con este contexto la función del display para generar los comandos y enviarlos es:

```

dfPlayer_Command(char Comando, char Feedback, char Param1, char Param2) {
    char Arreglo[10] = {0x7E, 0xFF, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xEF};

    Arreglo[3] = Comando;
    Arreglo[4] = Feedback;
    Arreglo[5] = Param1;
    Arreglo[6] = Param2; } Asigna las Variables al arreglo

int checksum = 0x1000; —> Inicializacion del checksum
for (int i=1; i<8; i++) checksum = checksum - Arreglo[i]; —> Calculo el checksum
Arreglo[7] = (char)(checksum << 8) —> Guardo el byte superior
Arreglo[8] = (char)(checksum) —> Restringe y guarda el byte inferior
if (dfPlayer_Send_Available() == 0) { —> Revisa si se puede enviar un mensaje, si se Puede
    UARTA

    dfPlayer_Send(Arreglo); —> no se incluye la longitud pues se modifica la funcion para que
    UARTA

    return 1;
} else return 0;
}

```