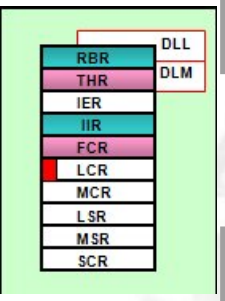
Recordando

# Resumen de los registros

LCR: Line Control Register – Base + 3 (activo alto)

| DLAB | SB | SP | EPS | PEN | STB | WLS1 | WLS0 |
|---|---|---|---|---|---|---|---|

LSR: Line Status Register – Base + 5 (activo alto)




| X | TEMT | THRE | BI | FE | PE | OE | DR |
|---|---|---|---|---|---|---|---|

IER: Interrupt Enable Register– Base + 1 (activo alto)

| 0 | 0 | 0 | 0 | MSI (Modem) | Rx line | THR (Tx) | RBR (Rx) |
|---|---|---|---|---|---|---|---|
| | | | | *Modem* | *err Rx* | *Tx* | *Rx* |

IIR: Interrupt Identification Register– Base + 2

| FIFO | | 0 | 0 | X | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|

Recordando

# Ejemplo Inicialización UART0

*Ejemplo: 9600,8,N,1*

| DLAB | SB | SP | EPS | PEN | STB | WLS1 | WLS0 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

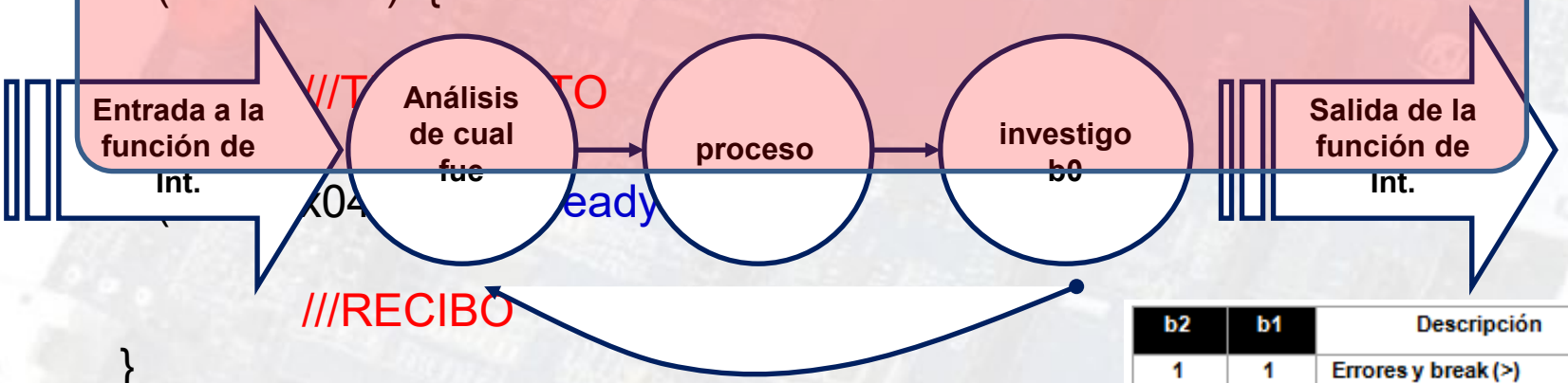
```
void InitUART0 (void)
{
    PCONP |= 0x01<<3;                  //1.- Registro PCONP (0x400FC0C4) – bit 3 en 1 prende la UART0
    PCLKSEL0 &= ~(0x03<<6);  //2.- Registro PCLKSEL0 - bits 6 y 7 en 0 seleccionan que el clk de la UART0 sea CCLK/4

    U0LCR = 0x00000083;      //3.- Registro U1LCR - Tx de 8 bits, 1 bit de stop, sin paridad, sin break cond, DLAB = 1
    U0DLM = 0x0A;            //4.- Registros U1DLL (0x40010000) y U1DLM (0x40010004) – 9600 baudios:
    U0DLL = 0x2C;            //0x00D9 p/115200;

//5.- Registros PINSEL0 (0x4002C000) y PINSEL1 (0x4002C004) - habilitan las funciones especiales de los pines:
    SetPINSEL(P0,2,PINSEL_FUNC1);      //TX0D ⇒ P0.2 ⇒ PINSEL0: 4/5
    SetPINSEL(P0,3,PINSEL_FUNC1);      //RX0D ⇒ P0.3 ⇒ PINSEL0: 6/7
    U0LCR = 0x03;                      //6.- Registro U1LCR, pongo DLAB en 0

    U0IER = 0x03;            //7. Habilito las interrupciones que correspondan (en la UART-IER- y en el NVIC-ISER)
    ISER0 |= (1<<5);         //ver pág. 77 del user manual
}
```

# Handler de comunicaciones

IIR: Interrupt Identification Register– Base + 2

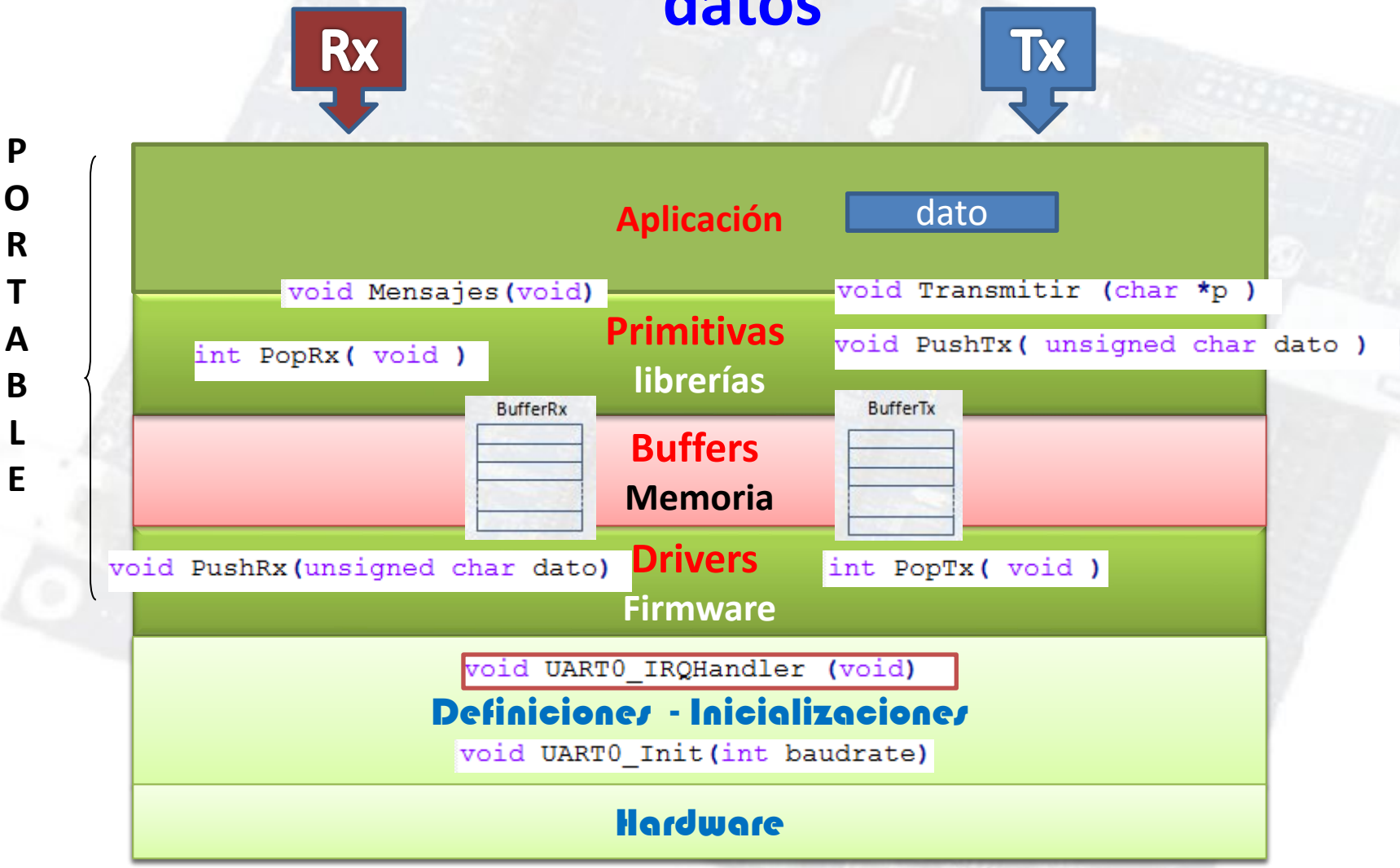
| FIFO | 0 | 0 | X | b2 | b1 | b0 |
|---|---|---|---|---|---|---|

```
void UART0_IRQHandler (void)  {
    uint8_t iir, dato;
    do  {
        iir = U0IIR; //IIR es reset por HW, una vez que lo leí se resetea.
        if ( iir & 0x02 )  {    //THRE
                ///T                TO
                x04                eady
                ///RECIBO
        }
        if ( iir & 0x06 )  { //errores

                ///Analízo ERRORES… ¿quién fue?

        }
    } while( ! ( iir & 0x01 ) ); /* me fijo si hay otra int. pendiente de atención: b0=1 (ocurre unicamente si dentro del mismo espacio temporal llegan dos interrupciones a la vez) */
}
```



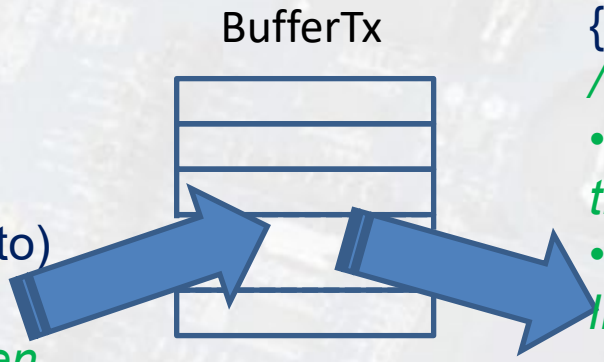

| b2 | b1 | Descripción |
|---|---|---|
| 1 | 1 | Errores y break (>) |
| 1 | 0 | Dato disponible |
| 0 | 1 | THR disponible |
| 0 | 0 | Estado del Modem (<) |

# El modelo de capas y el tratamiento de los datos

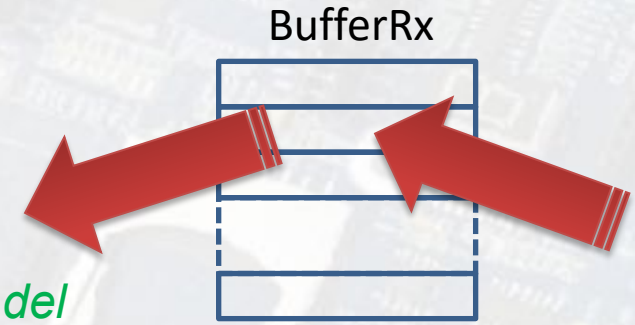


# Estrategias para la Tx y Rx

```
void  pushTx (uint8_t dato)
{
// primitiva que escribe en
// el buffer de Tx (BufferTx);}
```




```
int16_t  popTx (void)
{
/* FW:
• Extrae del BufferTx el dato a transmitir
• Incrementa índice y comprueba límite
• Devuelve el dato o  -1 por error
}
```



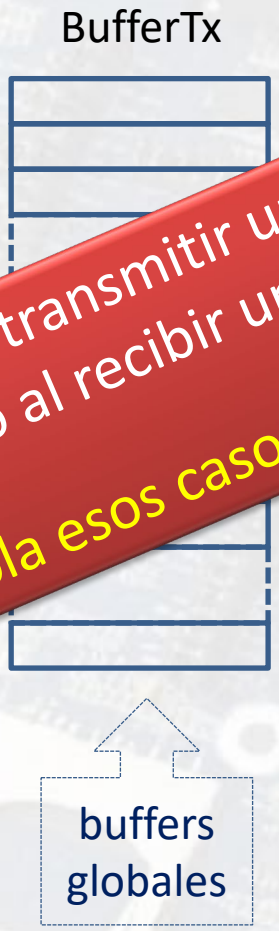

```
int16_t  popRx (void)
{
// primitiva que recibe del
// buffer de Rx (BufferRx)
// devolviendo  el dato o -1
por error
}
```

```
void  pushRx  (uint8_t dato)
{
/* FW:
• Introduce dato en BufferRx
• Incrementa índice y comprueba límite
•}
```

buffers globales

# Estrategias para la Tx y Rx

```
void pushTx (uint8_t dato)
{
// primitiva que escribe en
// el buffer de Tx (BufferTx);
// verificando TOPE e ini
// ciando la Tx si no
// transmisión
}

// recibe del
// de Rx (BufferRx)
// si los índices de Rx y Tx
// son diferentes (hay dato)
// verificando TOPE y
// devolviendo -1 por error
}
```




¿Y si BufferTx está vacío al transmitir un dato?
¿Y si BufferRx está lleno al recibir un dato?
¿Quién controla esos casos?

## 1ra. Aproximación a la isr

```
RT0_IRQHandler (void)

    iir, dato;


    iir = U0IIR;
    if (iir & 0x04) {   // data ready
        BufferRx[inxIn]=U0RBR;
        inxIn++;
        inxIn%=TOPE;
    }
    if (iir & 0x02) { // THRE disponible
        U0THRE=BufferTx[inxOut];
        inxOut++;
        inxOut%=TOPE;
     }
  } while ( !(iir & 0x01) ); // pendiente
}
```

# Estrategias para la Tx y Rx

```
void pushTx (uint8_t dato)
{
// primitiva que escribe en
// el buffer de Tx (BufferTx);
// verificando TOPE e ini-
// ciando la Tx si no hay
// transmisión en curso.
}

int16_t popRx (void)
{
// primitiva que recibe del
// buffer de Rx (BufferRx)
// si los índices de Rx y Tx
// son diferentes (hay dato)
// verificando TOPE y
// devolviendo -1 por error
}
```

BufferTx

BufferRx

buffers globales

**dato = popTx (dato);** ***//sabe usar el BufferTx***

**pushRx (dato);** ***//sabe usar el BufferRx***

```
void UART0_IRQHandler (void)
{
   uint8_t iir, dato;
   do
   {
      iir = U0IIR;
      if (iir & 0x04) {
            // data ready
            // --------------

      }
      if (iir & 0x02) {
            //

            // THRE --------------
       }
   } while ( !(iir & 0x01) ); // pendiente
}
```

| b2 | b1 | Descripción |
|---|---|---|
| 1 | 1 | Errores y break (>) |
| 1 | 0 | Dato disponible |
| 0 | 1 | THR disponible |
| 0 | 0 | Estado del Modem (<) |

# Handler de comunicaciones (P/Tx y Rx)

```
void UART1_IRQHandler (void)
{
    uint8_t iir,dato;

    int aux;
    do  {
        iir = U0IIR;

        if (iir & 0x04) { //Rx: bit 2:1 valen 10--> Interr por RX
            dato=U0RBR;
            push_RX(dato);
        }

        if (iir & 0x02) { //Tx: bit 2:1 valen 01--> Interr por TX
            aux = pop_TX();
            if(aux > 0)
                U0THR = aux;
            else
                TxStart = 0; //le aviso a la aplicación
                 //que puede volver a transmitir
        }
    }
    while( !( iir & 0x01 ) );
}
```

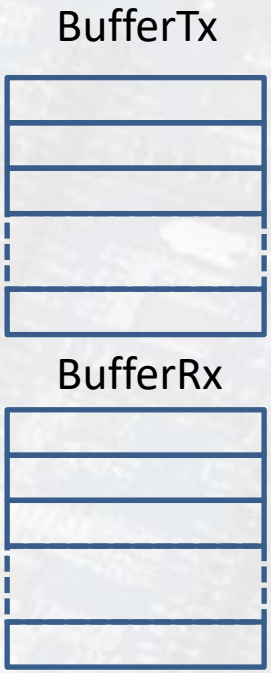
| b2 | b1 | Descripción |
|---|---|---|
| 1 | 1 | Errores y break (>) |
| 1 | 0 | Dato disponible |
| 0 | 1 | THR disponible |
| 0 | 0 | Estado del Modem (<) |

IIR: Interrupt Identification Register– Base + 2

| FIFO | 0 | 0 | X | b2 | b1 | b0 |
|---|---|---|---|---|---|---|

# Estrategias para la Tx y Rx: *Comprendiendo el mecanismo*

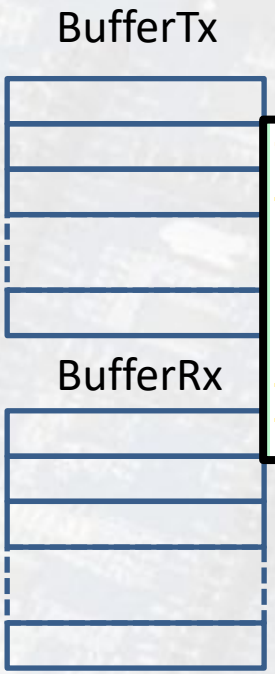
```
void pushTx (uint8_t dato)
{
// primitiva que escribe en
// el buffer de Tx (BufferTx);
// verificando TOPE e ini-
// ciando la Tx si no hay
// transmisión en curso.
}

int16_t popRx (void)
{
// primitiva que recibe del
// buffer de Rx (BufferRx)
// si los índices de Rx y Tx
// son diferentes (hay dato)
// verificando TOPE y
// devolviendo -1 por error
}
```




```
void UART1_IRQHandler (void)
{
    uint8_t iir,dato;

    int aux;
    do {
        iir = U0IIR;

        if (iir & 0x04) { //Rx: bit 2:1 valen 10--> Interr por RX
            dato=U0RBR;
            push_RX(dato);
        }

        if (iir & 0x02) { //Tx: bit 2:1 valen 01--> Interr por TX
            aux = pop_TX();
            if(aux > 0)
                U0THR = aux;
            else
                TxStart = 0; //le aviso a la aplicación
                 //que puede volver a transmitir
        }
    }
    while( !( iir & 0x01 ) );
}
```

# Estrategias para la Tx y Rx: *Operando sobre los buffers desde el FW*

```
void pushTx (uint8_t dato)
{
// primitiva que escribe en
// el buffer de Tx (BufferTx);
// verificando TOPE e ini-
// ciando la Tx si no hay
// transmisión en curso.
}

int16_t popRx (void)
{
// primitiva que recibe del
// buffer de Rx (BufferRx)
// si los índices de Rx y Tx
// son diferentes (hay dato)
// verificando TOPE y
// devolviendo -1 por error
}
```

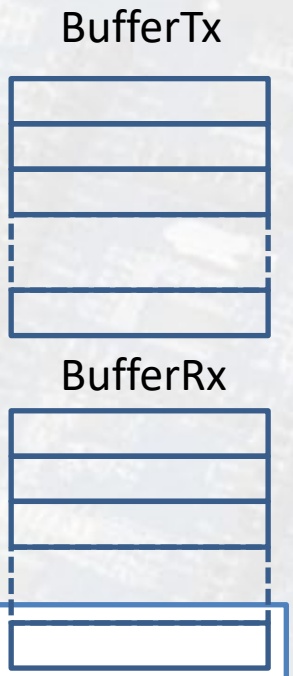



```
void push_RX(uint8_t dato)
{
        BufferRx[IndiceRxIn] = dato;
        IndiceRxIn++;
        IndiceRxIn %= TOPE_BUFFER_RX;
}
```

```
if (iir & 0x04) { //Rx: bit 2:1 valen 10--> Interr por RX
    dato=U0RBR;
    push_RX(dato);
}

if (iir & 0x02) { //Tx: bit 2:1 valen 01--> Interr por TX
    aux = pop_TX();
    if(aux > 0)
```

```
uint16_t pop_TX(void)
{
        int aux = -1; //si no hay que transmitir, devuelve -1
        if(IndiceRxIn != IndiceRxOut)
        {
                aux = BufferTx[IndiceRxOut];
                IndiceRxOut++;
                IndiceRxOut %= TOPE_BUFFER_TX;
        }
        return aux;
}
```

# Estrategias para la Tx y Rx: *Operando sobre los buffers desde las primitivas*




```
void pushTx( uint8_t dato)
{
	BufferTx[IndiceTxIn] = dato;

	IndiceTxIn ++;
	IndiceTxIn %= TOPE_BUFFER_TX;

	if ( TxStart == 0 )
	{
		TxStart = 1;
		ARRANCAR_TX;
	}
}
```

```
uint16_t popRx( void )
{
	uint16_t dato = -1;

	if ( IndiceRxIn != IndiceRxOut )
	{
		dato = (uint16_t) BufferRx[IndiceRxOut];
		IndiceRxOut ++;
		IndiceRxOut %= TOPE_BUFFER_RX;
	}
	return dato;
}
```

```
void UART1_IRQHandler (void)
{
    uint8_t iir,dato;

    int aux;
    do {
        iir = U0IIR;

        if (iir & 0x04) { //Rx: bit 2:1 valen 10--> Interr por RX
            dato=U0RBR;
            push_RX(dato);
        }

        if (iir & 0x02) { //Tx: bit 2:1 valen 01--> Interr por TX
            aux = pop_TX();
            if(aux > 0)
                U0THR = aux;
            else
                TxStart = 0; //le aviso a la aplicación
                 //que puede volver a transmitir
        }
    }
    while( !( iir & 0x01 ) );
}
```

```
#define ARRANCAR_TX ( U0THR = BufferTx[IndiceTxOut] )
```

# ¿y si quiero mandar una string?

```
void pushTx(uint8_t dato)
{
	BufferTx[IndiceTxIn] = dato;

	IndiceTxIn ++;
	IndiceTxIn %= TOPE_BUFFER_TX;

	if ( TxStart == 0 )
	{
		TxStart = 1;
		ARRANCAR_TX;
	}
```

```
void EnviarString (const char *str)
{
    uint32_t i;
    for( i = 0 ; str[i] ; i++ )
        PushTx( str[i] );

}
```

```
#define ARRANCAR_TX ( UOTHR = BufferTx[IndiceTxOut] )
```

# Resumiendo

```
void  pushTx (uint8_t dato)
{
// primitiva que escribe en
// el buffer de Tx (BufferTx);
// verificando TOPE e ini-
// ciando la Tx si no hay
// transmisión en curso.
}
```

BufferTx

```
int16_t  popTx (uint8_t dato)
{
/* FW:
• Extrae del BufferTx el dato a transmitir
• Incrementa índice y comprueba límite
• Devuelve el dato o  -1 por error
}
```

```
int16_t  popRx (void)
{
// primitiva que recibe del
// buffer de Rx (BufferRx)
// si los índices de Rx y Tx
// son diferentes (hay dato)
// verificando TOPE y
// devolviendo -1 por error
}
```

BufferRx

```
void  pushRx (uint8_t dato)
{
/* FW:
• Introduce dato en BufferRx
• Incrementa índice y comprueba límite
}
```

# Agradecimientos

Ing. Marcelo Trujillo

Ing. Marcelo Giura

Ing. Gabriel Socodato

Ing. Marcelo Romeo

# Ejercicio 1

- Se pide diseñar la máquina de estado que atienda la recepción de una trama de datos a través del puerto serie.
- La trama comienza con el símbolo * y finaliza con el símbolo #.  Recibe 4 bytes de datos.
- Qué elementos requiere para el diseño?
- Qué estados se definen? Cuales son?
- Qué eventos y acciones aplicamos?
- Puede formar parte de otra máquina? De cual?

# Ejercicio 2

- Tomando de base el ejercicio anterior se recibe ahora una trama que comienza con * y finaliza con # pero no se sabe cuantos datos contiene.
- Se conoce que el primer dato que llega corresponde a la parte baja de un valor de “id” de usuario y el siguiente con la parte alta.
- Se pide validar la trama e incluir los cambios que considere necesarios al modelo anteriormente desarrollado.