Telégrafo

Sistemas Embebidos

Gutiérrez, Juan Lucas

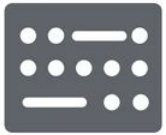
¿Qué es un Telégrafo?

Es un sistema de comunicación capaz de enviar un mensaje a distancia, a través de cables u ondas.

Tele (distancia) + Graphos (escritura)

Interfaz de comunicación

Codificación



Morse Code

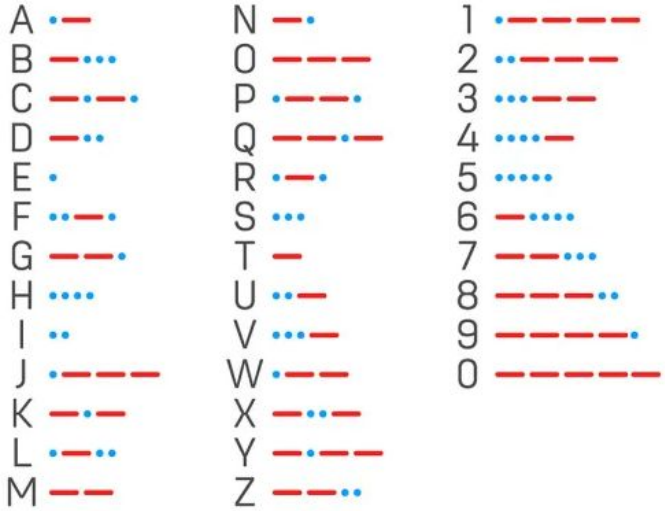
¿Qué es el código Morse?

Es un sistema de codificación alfanumérico, que traduce letras y números a una sucesión de pulsos.

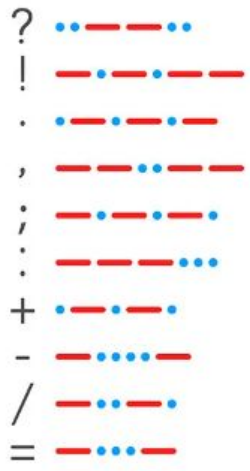
Un pulso puede ser eléctrico, lumínico, sonoro, etc.

¿Que es el código Morse?

Cada caracter se representa mediante una sucesión de puntos (“dits”) y rayas (“dahs”)



basico



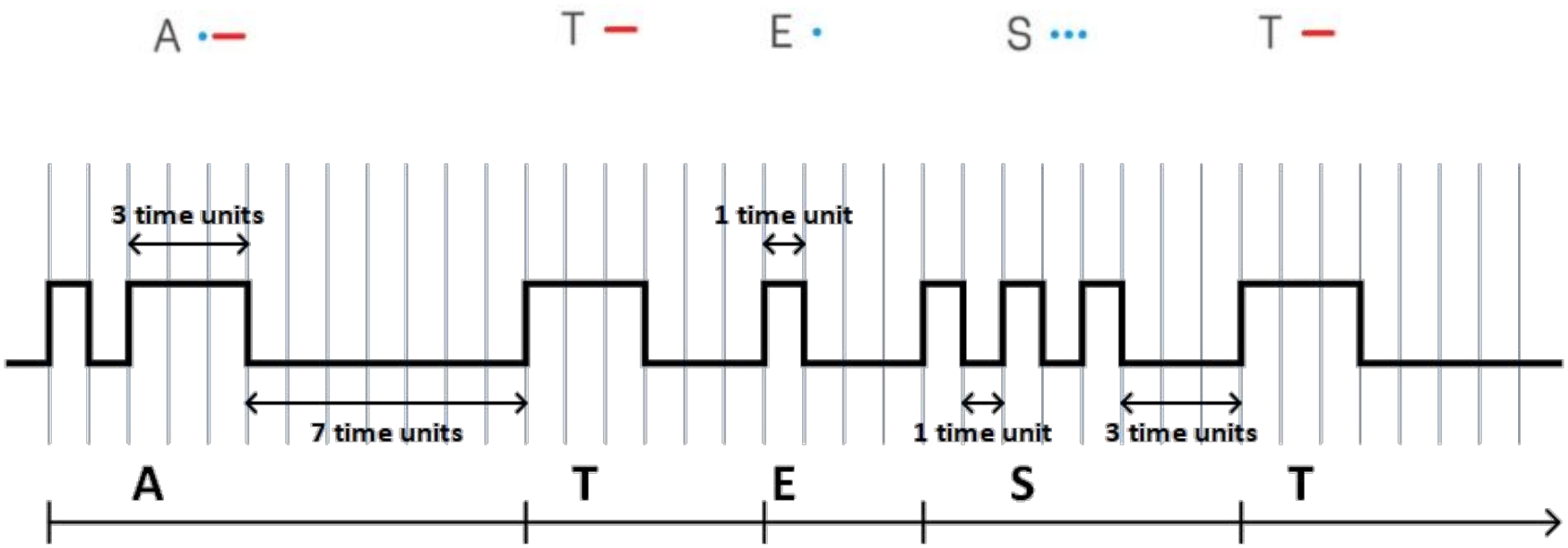
extendido

¿Que es el código Morse?

- En qué se diferencia un punto de una raya?
- Si los caracteres tienen “tamaños” distintos, cómo identificamos cuando termina un caracter?
- Cuándo termina una palabra?

¿Que es el código Morse?

Codificación temporal: las tramas constan de símbolos y tiempos!



Implementación

Requerimientos:

SOFTWARE

- Detección de pulsos
- Detección de tiempos entre pulsos
- Decodificación/validación de tramas
- Comunicación por puerto serie
  - Configuración
  - Visualización sincrónica y asincrónica

HARDWARE

- Generación de señal adecuada
- Indicación de señal
  - Visual
  - Sonora

Implementación: Software

Máquina de estados en esquema foreground-background / event-triggered

Background:

```
12  int main(void){
13
14      init_system();
15      morse_init(&morse, 130);
16      enable_print_menu();
17
18      while (1){
19          morse_run(&morse);
20      }
21  }
```

```
110 void morse_run(morse_s* self){
111     if(morse_fsm_get_event(self) != NO_EVENT){
112         morse_fsm_switch(self);
113     }
114
115     if(is_print_menu_enabled() == true){
116         print_menu();
117     }
118 }
119
```

Implementación: Software

Máquina de estados en esquema foreground-background / event-triggered

Foreground:

```
12  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
13      HAL_EXTI_ClearPending(GPIO_Pin, EXTI_TRIGGER_RISING_FALLING);
14      if(GPIO_Pin == GPIO_PIN_9){
15          if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_9) == GPIO_PIN_RESET){
16              turn_led_on();
17              js_set_signal_state(SIGNAL_LOW);
18          }else{
19              turn_led_off();
20              js_set_signal_state(SIGNAL_HIGH);
21          }
22      }
23  }
```

```
7   void js_set_signal_state(button_state_t state){ js.button.state = state; }
```

Implementación: Software

Máquina de estados en esquema foreground-background / event-triggered

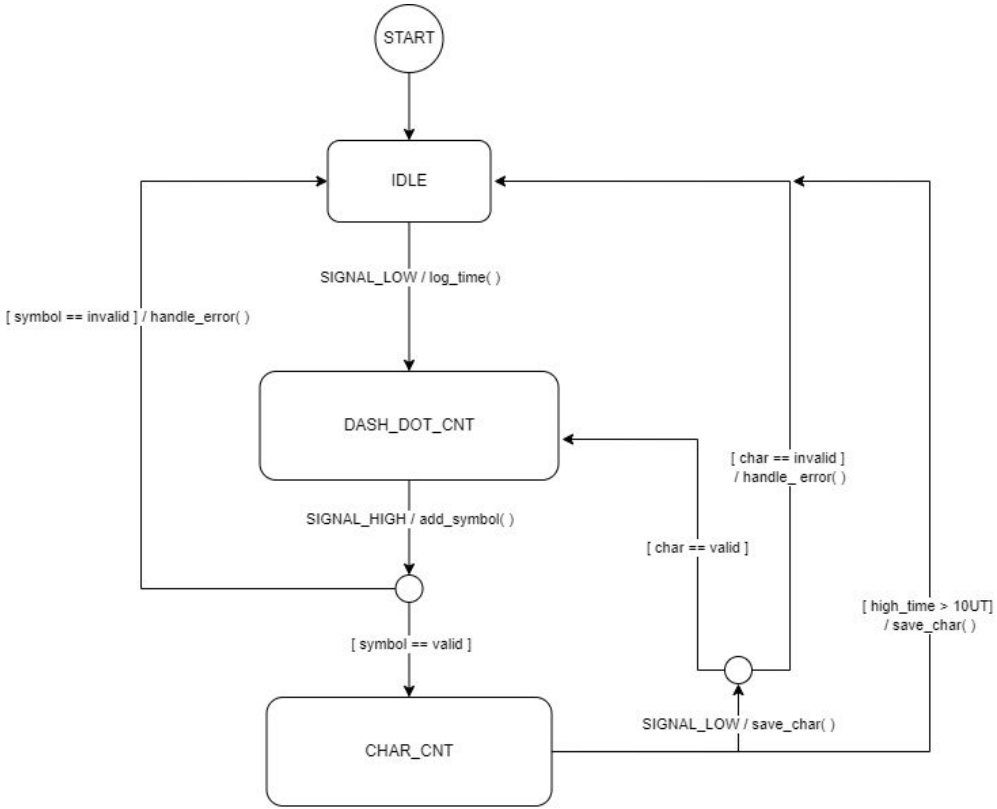
ESTADOS

- SYM_CNT: decodifica símbolos según tiempo de la señal “presionada”
- CHAR_CNT: decodifica secuencias según tiempo “libre” de la señal
- IDLE: cambia estados según eventos

EVENTOS

- PRESSED: señal “presionada”
- RELEASED: señal “liberada”

Implementación: Software



Implementación: Codificación de señal

Cómo agregar cada símbolo en una estructura para poder validarlo posteriormente de forma **simple** y eficiente?

Implementación: Codificación de señal

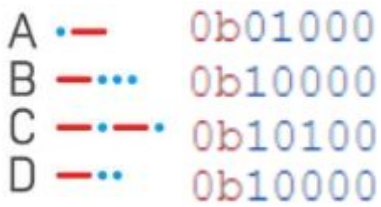
Cómo agregar cada símbolo en una estructura para poder validarlo posteriormente de forma **simple** y eficiente?

- Mediante vector char
  - Cargar símbolo como “**.**” o “**-**”
  - Generar un diccionario de vectores char
    para cada caracter alfanumérico
  - Verificar si el vector entrante coincide con
    alguno y luego derivar de qué caracter se trata

Implementación: Codificación de señal

Cómo agregar cada símbolo en una estructura para poder validarlo posteriormente de forma simple y **eficiente**?

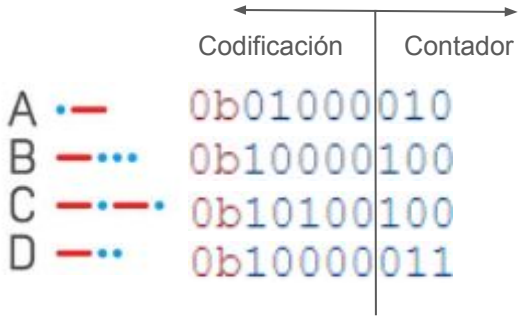
- Mediante codificación “custom”
  - Cargar cada símbolo en variable uint8 bit a bit



Implementación: Codificación de señal

Cómo agregar cada símbolo en una estructura para poder validarlo posteriormente de forma simple y **eficiente**?

- Mediante codificación “custom”
  - Cargar cada símbolo en variable uint8 bit a bit



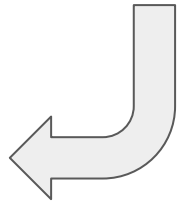
Implementación: Codificación de señal

Codificación “custom”

```
327 morse_status_t morse_add_symbol(morse_s* self, morse_sym_t sym){
328     static uint8_t symbol_position_mask = START_POSITION;
329     // start position == 0b10000000
330         if(sym == DOT){
331             // self->bin_char &= ~(self->symbol_counter);
332         }else if(sym == DASH){
333             self->bin_char |= (symbol_position_mask >> self->symbol_counter);
334         }else{
335             self->status = ERR_CHAR_UNKNOWN;
336             return FAIL;
337         }
338         self->symbol_counter++;
339         return OK;
```

- variable uint8_t = 0b00000000
- Se agrega simbolos de izquierda a derecha, uno por bit
- “.” -> 0 “-” -> 1
- Max 5 bits
- Cada símbolo suma uno en los 3 LSb

```
362 morse_status_t morse_add_char(morse_s* self){
363     morse_status_t ret;
364
365     self->bin_char |= self->symbol_counter;
```



Implementación: Codificación de señal

Codificación “custom”

```
466     switch(bin_char){
467         case(0b01000010):
468             return('A');
469         case(0b10000100):
470             return('B');
471         case(0b10100100):
472             return('C');
473         case(0b10000011):
474             return('D');
475         case(0b00000001):
476             return('E');
            ...
539         default:
540             return 0;
```

SWITCH

- case(codificación): caracter
- default: error
    - Ventaja: simple, fácil de escalar y debuguear
    - Desventaja: más “lento” (!)

Implementación: Codificación de señal

Codificación “custom”

```
446  #ifdef MORSE_DECODE_USING_LUT
447      const static char lut[256] =
448      {0,'E','I','S','H','5',0,0,0,0,0,0,0,'6',0,0,
449       0,0,0,0,'B',0,0,0,0,0,0,0,0,'7',0,0,
450       0,0,0,'D','L',0,0,0,0,0,0,0,0,0,0,0,
451       0,0,0,0,'Z',0,0,0,0,0,0,0,0,'8',0,0,
452       0,0,'N','R','F',0,0,0,0,0,0,0,0,0,0,0,
453       0,0,0,0,'C',0,0,0,0,0,0,0,0,0,0,0,
454       0,0,0,'G','P',0,0,0,0,0,0,0,0,0,0,0,
455       0,0,0,0,0,0,0,0,0,0,0,0,0,'9',0,0,
456       0,'T','A','U','V','4',0,0,0,0,0,0,0,0,0,0,
457       0,0,0,0,'X',0,0,0,0,0,0,0,0,0,0,0,
458       0,0,0,'K',0,0,0,0,0,0,0,0,0,0,0,0,
459       0,0,0,0,'Q',0,0,0,0,0,0,0,0,0,0,0,
460       0,0,'M','W',0,'3',0,0,0,0,0,0,0,0,0,0,
461       0,0,0,0,'Y',0,0,0,0,0,0,0,0,0,0,0,
462       0,0,0,'o','J','2',0,0,0,0,0,0,0,0,0,0,
463       0,0,0,0,0,'1',0,0,0,0,0,0,0,'0',0,0};
464      return lut[bin_char];
```

LUT

- Vector char
- Tamaño -> máximo de la codificación: 8 bits = 256
- vector[codificación] = caracter
- vector[codificación] != error
  - Ventaja: velocidad
  - Desventaja: espacio en memoria, difícil de debuguear y escalar

Implementación: Visualización y configuración

Puerto serie (8N1 b:115200):

Configuración:

- Se ingresan comandos por terminal
- Se capturan mediante interrupción por recepción serie
- Comandos:
  - ST*n*: Set unit Time: se valida n y se configura como el nuevo unit time
  - GT: Get unit Time: muestra el unit time configurado
  - CB: Clear Buffer: limpia el buffer de mensajes
  - SB: Show Buffer: muestra el buffer de mensajes

Implementación: Visualizacion y configuración

Puerto serie (8N1 b:115200):

Configuración:

```
Telegraph
Initializing System
Init morse object

Last Word:
Telegraph Menu
--------------
GT: Get Morse Unit Time
STxxx: Set Morse Unit Time to xxx miliseconds
CB: Clear word buffer
SB: Show word buffer
```

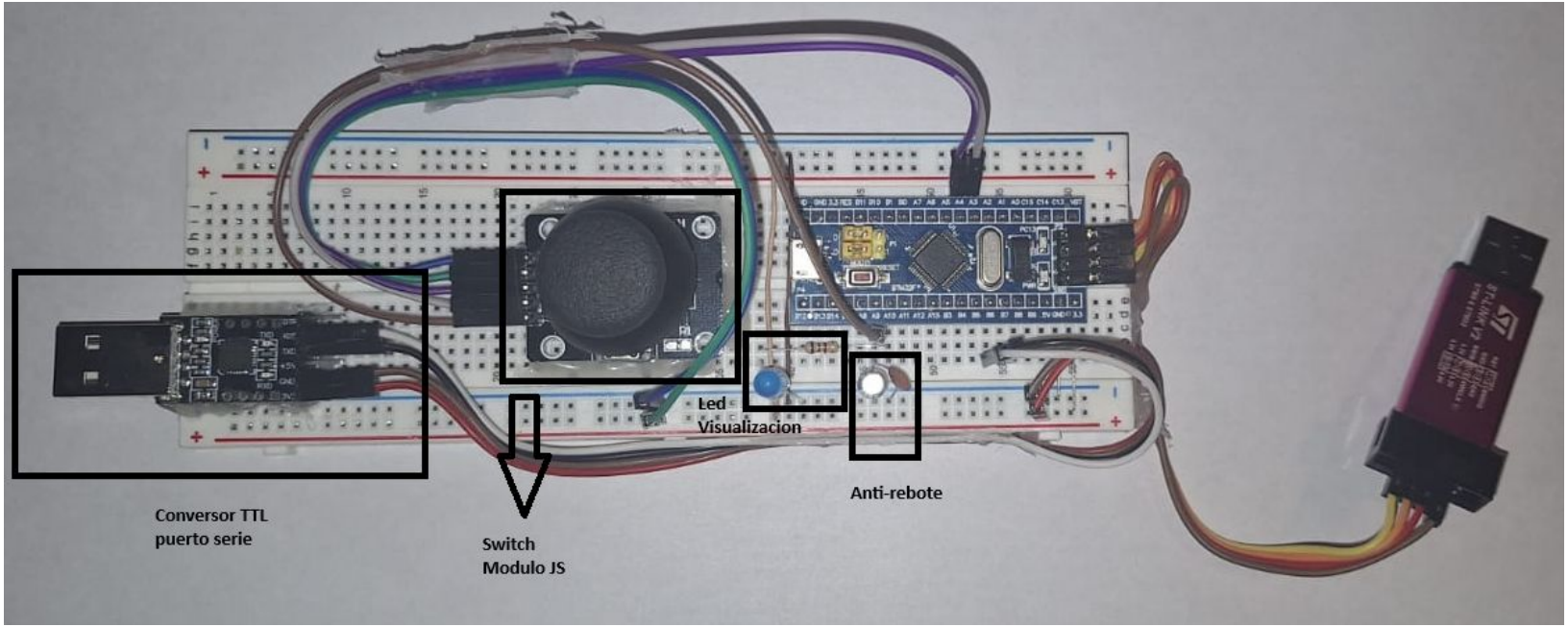
Implementación: Visualizacion y configuración

Puerto serie (8N1 b:115200):

Visualización:

Implementación: Hardware

Protoboard para ensayo:



Resultados:

Señal vista desde PulseView: