Laboratorio de Microprocesadores

Trabajo Práctico №2

ITBA

Máspero Martina
Mestanza Joaquín
Nowik Ariel
Regueira Marcelo

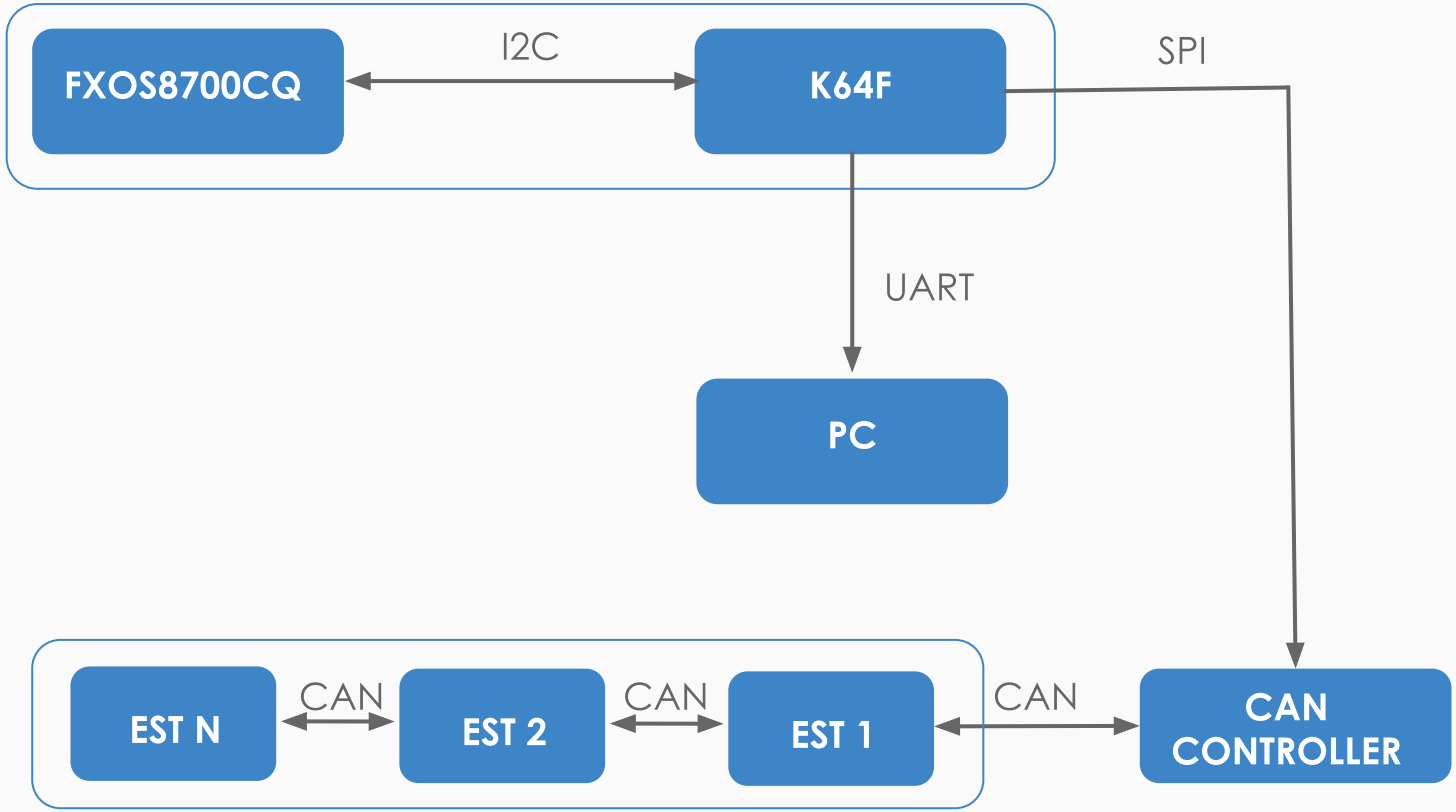


Diagrama de comunicaciones

Capas e interfaces

Application

App ExternalManager Posicionamiento App PC

Hardware Abstraction Layer

UART SPI CAN I2C Sensores Timer

MicroController Abstraction Layer

Systick GPIO

UART

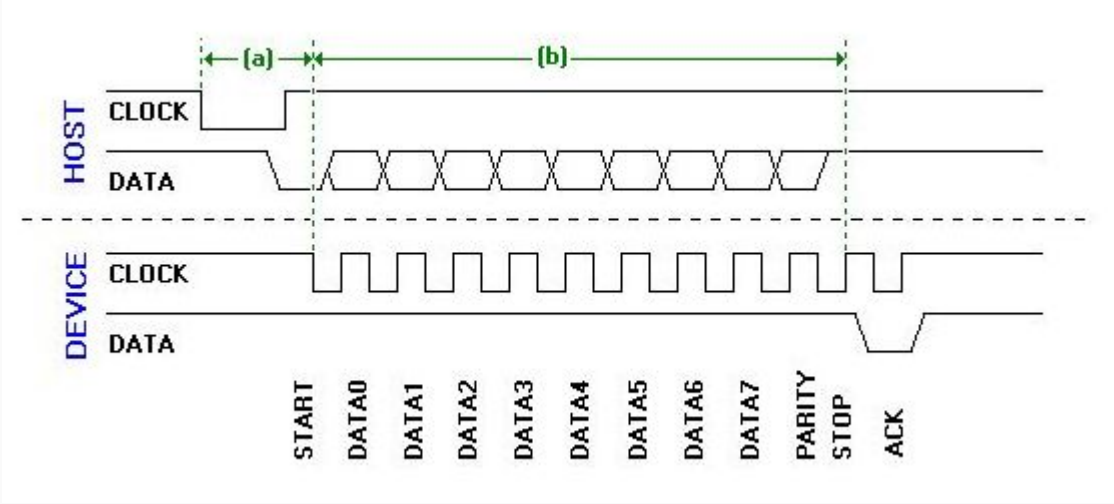
Para poder comunicarse desde la placa kinetis hacia la computadora utilizamos protocolo UART.
Con la función sendWord se puede enviar un string

UART.h

```
void uartInit(uart_cfg_t config);

void sendWord(const char *word);
```

UART

Implementamos solo el protocolo para enviar datos de la kinetis a la PC. Usamos interrupciones de UART.



SPI

Para poder comunicarnos con CAN Controller with integrated Transciever, tuvimos que setear el clock de SPI de una manera especial, los detalles se encuentran en MSTRCFG.

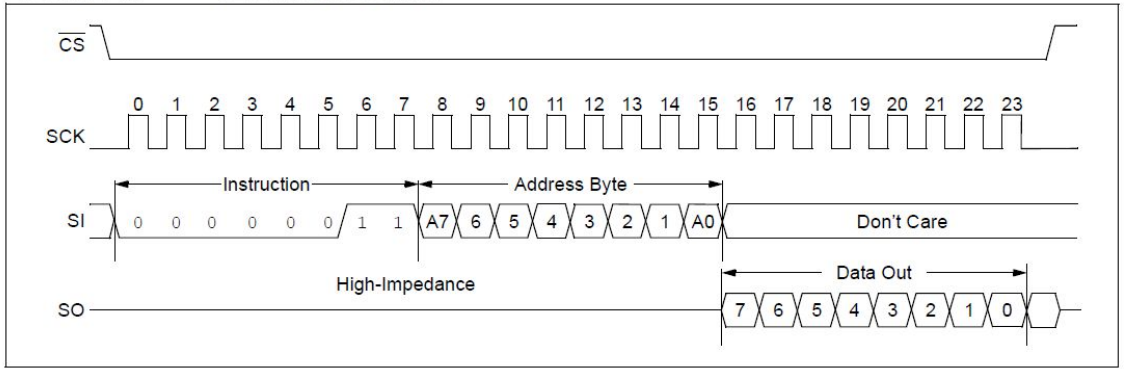
SPI.h

```
void MSTRCFG(MSTRCFG_t * master_cfg);

void masterInitialize(uint8_t SPI_n);

void SPI_Initialize(void(*funcallback)(void));

void getNotLastCommand(spi_command * com);

uint16_t masterWriteReadByte(uint8_t SPI_n,
spi_command* com, uint16_t data);

void masterWriteReadBuffer(uint16_t *data,
uint32_t size, uint_16* received_data, uint8_t
SPI_n);
```

SPI

Cada instrucción de la placa CAN tiene una determinada forma de mandar los datos por la interfaz SPI. Dejamos un ejemplo a modo de demostración.

FIGURE 5-2: READ INSTRUCTION



Dado este formato, se tuvo que implementar una función BufferWriteRead, ya que en un momento estamos escribiendo y en otro estamos leyendo. Los diferentes servicios de CAN se encargan tomar o ignorar la información recibida.

CAN

El protocolo CAN implementado se abstrajo a nivel de inicialización, escritura mediante arreglo de bytes (máximo 8), y recepción, donde se obtienen los 8 Bytes del buffer y la cantidad recibida (para poder saber a cuantos tener en cuenta).

CAN.h

```
void init_CAN(int ID, void (*funcallback)(void));

void send_CAN(int ID, char * buffer, int bufflen);

RXB_RAWDATA_t getRXB_Data_CAN(void);

bool getTXFlag_CAN(void);
```

I2C

```
typedef struct
{
        uint8_t * data;
        uint8_t data_size;
        uint8_t register_address;
        uint8_t slave_address;
        pfunc callback;
        I2C_FAULT fault;
}I2C_COM_CONTROL;
```

I2C.h

void **i2cInit (uint8_t channel);**

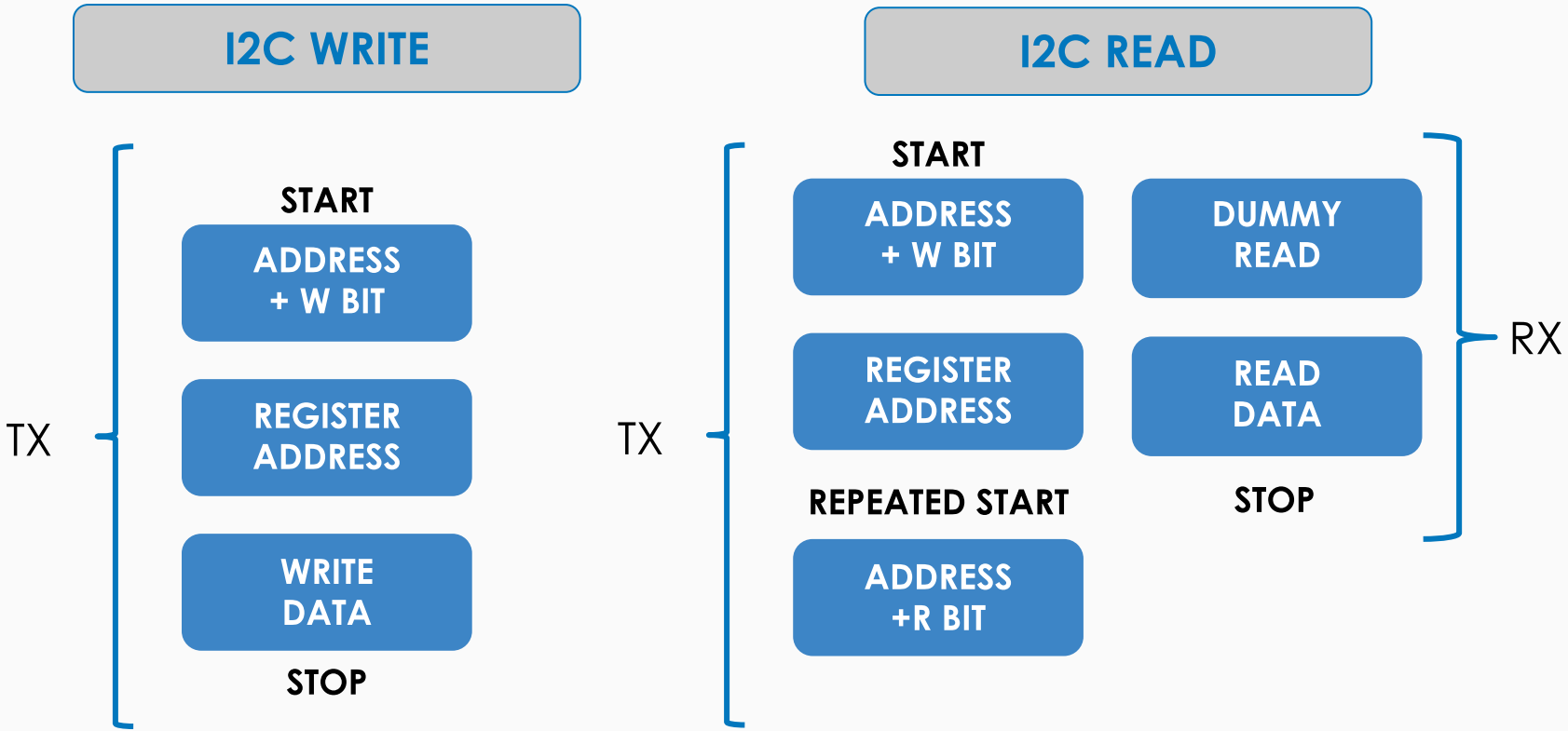
void **i2cReadMsg(**I2C_COM_CONTROL* **i2c_com);**

void **i2cWriteMsg(**I2C_COM_CONTROL * **i2c_com);**

I2C_FAULT **i2cReadMsgBlocking (**uint8_t * **buffer,** uint8_t **data_size,** uint8_t **register_address,** uint8_t **slave_address);**

I2C_FAULT **i2cWriteMsgBlocking (**uint8_t * **msg,** uint8_t **data_size,** uint8_t **register_address,** uint8_t **slave_address);**

I2C STAGES


I2C WRITE
TX
START
ADDRESS + W BIT
REGISTER ADDRESS
WRITE DATA
STOP
I2C READ
TX
START
ADDRESS + W BIT
REGISTER ADDRESS
REPEATED START
ADDRESS +R BIT
DUMMY READ
READ DATA
STOP
RX

Sensores

```
typedef struct
{
        SRAWDATA * pMagnData;
        SRAWDATA * pAccelData;
        callbackp callback;
        I2C_FAIL error;

}read_data;
```

Sensores.h

```
I2C_FAIL _mqx_ints_FXOS8700CQ_start(void);

void ReadAccelMagnData(read_data * data);
```

Posicionamiento

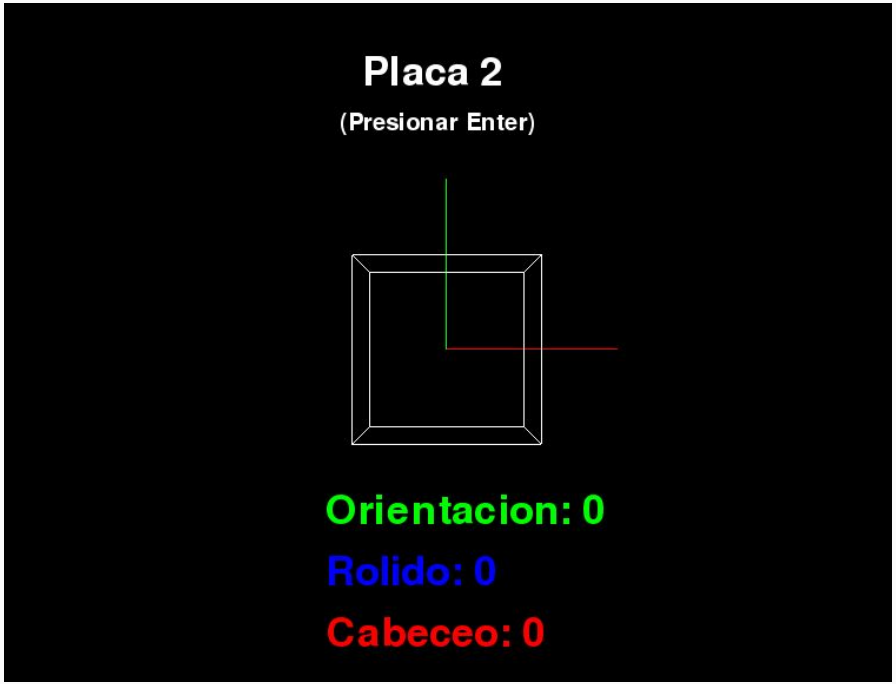
Posicionamiento toma la data cruda del acelerómetro y del magnetómetro y hace la transformación a rolido, cabeceo y orientación.

Posicionamiento.h

```
void Position_InitDrv(void (*funcallback)(void));
void Position_Update(void);
roll_t Position_GetRoll(void);
pitching_t Position_GetPitch(void);
orientation_t Position_GetOrientation(void);
int Position_GetChangeEvent(void);
```

OpenGL

Utilizamos la libería pyOpenGL para graficar la posición de las 8 placas. Con la función glRotatef() rotamos los ejes, y con la función glTranslatef() cambiamos de lugar los objetos.
Se debe presionar Enter para actualizar la posición de las placas, ya que se debe resetear la posición al inicio al rotar (de lo contrario hubieran sido necesarios quaterniones)



External Manager

Es el encargado de enviar la data al resto de las estaciones utilizando CAN y a la PC utilizando UART.

ExternalManager.h

void ExternManager_init(uint8_t group_num);

void ExternManager_send2Ext(BoardParams_t myBoard, uint8_t typeUPD);

¿ Preguntas ?