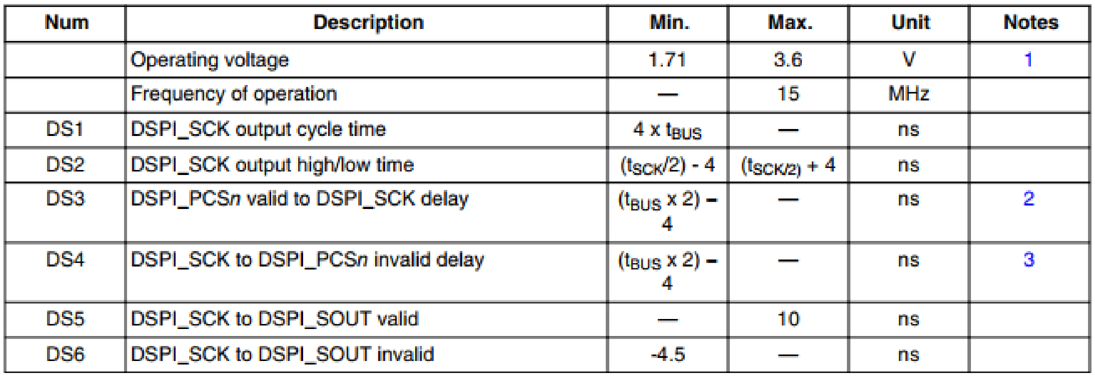
Embedded Firmware

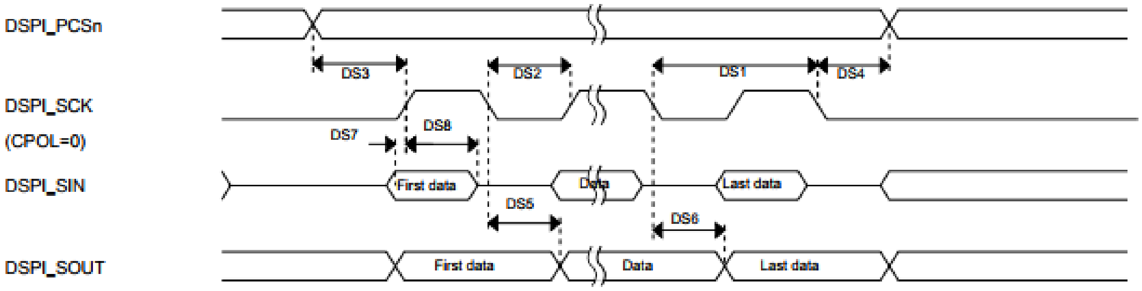
We expect the basic functionality in this test to take around 3 hours, but as it's an open ended test it could take longer to complete. If you haven't finished it after 3 hours, you can still send it in unfinished or you can take more time to do it, it's up to you.

In robotics and motor control position sensing is very important, your task is to create a driver for a specified high speed position sensor. Our target device is a ARM Cortex M0 microcontroller. It has an SPI peripheral with the following specifications:



Num	Description	Min.	Max.	Unit	Notes
	Operating voltage	1.71	3.6	V	1
	Frequency of operation	—	15	MHz	
DS1	DSPI_SCK output cycle time	4 x t_{BUS}	—	ns	
DS2	DSPI_SCK output high/low time	$(t_{SCK}/2) - 4$	$(t_{SCK/2}) + 4$	ns	
DS3	DSPI_PCS*n* valid to DSPI_SCK delay	$(t_{BUS} \times 2) - 4$	—	ns	2
DS4	DSPI_SCK to DSPI_PCS*n* invalid delay	$(t_{BUS} \times 2) - 4$	—	ns	3
DS5	DSPI_SCK to DSPI_SOUT valid	—	10	ns	
DS6	DSPI_SCK to DSPI_SOUT invalid	-4.5	—	ns	

***Note**: f_{BUS} is fixed at 60Mhz*



1. Using the attached header file and data above, create a simple C module which mocks the `SPI_configureTransfer` and `SPI_transferBlocking` functions.
   Hint: The mock only needs to verify inputs and return relevant values.

2. In C, write a device driver for a AMS AS5047D High Speed Position Sensor. Use the attached header file to communicate via SPI and the AMS AS5047D datasheet.
   Hint: Your mock module may come in handy.

The driver should allow the configuration and reading of the chip's registers. Because this is a complex chip with a number of registers and functions, we are only looking for a driver which:

- Can easily set the SETTINGS1 and SETTINGS2 registers
- Can read errors from the error register (ERRFL)

- Can read the compensated encoder angle register (ANGLECOM)
    - In both raw values (a 14 bit number) and in degrees

This chip has the ability to program settings into non-volatile memory. The driver does not need to use this functionality.

Use any IDE/compiler/tools you are comfortable with, document your choice along with any configurations or flags. We are looking for best practices, clean code and tests. Submit all code via a git repository, if possible private, with clear instructions for build, usage, etc. Don't hesitate to send us an email if you have any questions.

```c
#ifndef _AUTOMATA_SPI_H
#define _AUTOMATA_SPI_H

#include <stdint.h>
#include <stdbool.h>

/* SPI transfer return values */
typedef enum _SPI_RETURN {
  SPI_RETURN_ok = 0,        /* OK/success */
  SPI_RETURN_nullptr,       /* Argument contains a null pointer */
  SPI_RETURN_invalid_arg    /* Argument out of range/invalid */
} SPI_RETURN_T;

typedef enum _POLARITY {
  SPI_ACTIVE_LOW = 0,       /* Signal active low*/
  SPI_ACTIVE_HIGH           /* Signal active high */
} SPI_POLARITY_T;

typedef enum _CPOL {
  SPI_IDLE_LOW = 0,         /* CLK idle low */
  SPI_IDLE_HIGH             /* CLK idle high */
} SPI_CPOL_T;

typedef enum _CHPA {
  SPI_EDGE_RISING = 0,      /* Data latch on rising falling */
  SPI_EDGE_FALLING          /* Data latch on falling rising */
} SPI_CHPA_T;

/* SPI transfer configuration structure */
typedef struct _SPI_CONFIG {
  SPI_POLARITY_T cs_pol;    /* Chip select polarity */
  SPI_CPOL_T cpol;          /* CPOL */
  SPI_CHPA_T cpha;          /* CHPA */
  uint8_t bits_per_frame;   /* Bits per frame, min 4 - max 16 */
  uint32_t baudrate;        /* Transfer baud rate (bits/second) */
  uint32_t cs_to_sck;       /* Time (ns) between asserting CS & SCK edge */
  uint32_t sck_to_cs;       /* Time (ns) between SCK edge & releasing CS */
} SPI_CONFIG_T;

/* SPI transfer structure */
typedef struct _SPI_TRANSFER {
  uint8_t *tx_data;          /* Send buffer. */
  uint8_t *rx_data;          /* Receive buffer. */
  uint32_t number_of_bytes;  /* Transfer size in bytes */
} SPI_TRANSFER_T;

/* Configure the SPI transfer */
SPI_RETURN_T SPI_configureTransfer(const SPI_CONFIG_T *config);
```

```
/* SPI transfer */
SPI_RETURN_T SPI_transferBlocking(const SPI_TRANSFER_T *transfer);

#endif /* _AUTOMATA_SPI_H */
```