SPI:-

1)brief SPI\_init() :-

\* Function to initialize the SPI

\* SPI\_configure() is being called to configure the required GPIO pins

\* The SPI is then enabled and set as the Master

\* The baud rate prescale and baud rate divisor is then being set

2)@brief SPI\_state()

\* Function that shows the state of SPI

3)@brief SPI\_write\_byte(uint8\_t byte)

\* Function to write byte to the SPI

\* It writes the byte if the transmitter buffer is empty

4)@brief SPI\_send\_packet(uint8\_t\* p, uint8\_t length)

\* Function to send packet

\* To check if the transmit buffer is empty and then sends the data for given

\* length

5) @brief SPI\_read\_byte()

\* Function to read the SPI

\* SPI\_configure() is being called to configure the required GPIO pins

\* The SPI is then enabled and set as the Master

\* The baud rate prescale and baud rate divisor is then being set

6) @brief SPI\_flush()

\* Function to flush the transmit and receive buffers

NORDIC:-

1) @brief nrf\_read\_register

\* It reads the SPI\_write\_byte register in a variable and returns the value

2) @brief nrf\_write\_register

\* It sets the SPI\_write\_byte register to a value

3) @brief nrf\_read\_status

\* It is a function to read the nordic status register

\* SPI\_write\_byte has been set to FF and a variable is used to store the result

\* for SPI\_read

4) @brief nrf\_write\_config

\* GPIO is low

\* It is a function to write command to the Config register

\* Data is written to config register and that data is being stored in

\* config variable. The GPIO is then set to high

5) @brief nrf\_read\_config

\* The Clock to Port B is being enabled.

\* GPIO is low

\* It is a function to read command to the Config register

\* A dummy value is sent to receive data

\* The GPIO is then set to high and the dummy value is being returned

6) @brief nrf\_read\_rf\_setup

\* Function to read the rf setup register

\* GPIO is low

\* nrf-write-register is used for NORDIC\_RF\_SETUP\_REG

\* A dummy value is sent to receive data

\* The GPIO is then set to high

7) @brief nrf\_write\_rf\_setup

\* Function to write the rf setup register

\* GPIO is low

\* nrf-write-register is used for NORDIC\_RF\_SETUP\_REG

\* A value is written

\* The GPIO is then set to high

8) @brief nrf\_read\_rf\_ch

\* Function to read the ch register

\* GPIO is low

\* nrf-write-register is used for NORDIC\_RF\_CH\_REG

\* A value is written

\* The GPIO is then set to high

\* The fifo status is then read and returned

9) @brief nrf\_write\_rf\_ch

\* Function to write to the ch register

\* GPIO is low

\* nrf-write-register is used for NORDIC\_RF\_CH\_REG

\* values are written on the channel

\* The GPIO is then set to high

10) @brief nrf\_read\_TX\_addr

\* Function to read the TX\_ADDR

\* GPIO is low

\* nrf-read-register is used for reading the TX\_ADDR register

\* values are written using SPI\_write\_byte

\* The GPIO is then set to high

11) @brief nrf\_write\_TX\_ADDR

\* Function to write to the tx address register

\* GPIO is low

\* nrf\_write\_register is used for TX\_ADDR register

\* values are written using SPI\_write\_byte

\* The GPIO is then set to high

12) @brief nrf\_read\_fifo\_status

\* Function to read the nrf\_fifo\_status

\* GPIO is low

\* nrf\_read\_register is used for FIFO\_STATUS\_REG

\* values are written

\* The GPIO is then set to high

13) @brief nrf\_flush\_tx\_fifo

\* Function to flush\_tx\_fifo

\* GPIO is low

\* SPI\_write\_byte sending 0xE1

\* The GPIO is then set to high

14) @brief nrf\_flush\_rx\_fifo

\* Function to flush\_rx\_fifo

\* GPIO is low

\* SPI\_write\_byte sending 0xE2

\* The GPIO is then set to high

RTC:-

1) @brief date\_to\_sec()

\* Function that is used to convert date to

\* seconds

2) @brief RTC\_configure()

\* Function to configure the RTC

3) @brief RTC\_seconds\_IRQHandler()

\* An IRQ handler that is used to handle

\* the interrupt given by RTC

DMA:-

1)

@brief memmove\_dma

\* Function that is used for moving the data from source to destination using DMA

\* The clock is being enabled for DMA and DMA\_MUX

\* Source and destination addresses are being given to Source address and destination address

\* registers respectively

\* overlap is being handled and then data is being transfered for 1, 2 or 4 bytes

\* The interrupt is enabled after transfer and then DMA is started using DMA mask

2) @brief memmset\_dma

\* Function that is used for setting value to the source using dma

\* The clock is being enabled for DMA and DMA\_MUX

\* Source and destination addresses are being given to Source address and destination address

\* registers respectively

\* memset is being done for 1 or 2 or 4 bytes

\* Interrupt is enabled after completion of memset

\* The channel 0 is enabled

\* The DMA is being started using the DMA start mask

3) @brief DMA\_overalp\_handler

\* Function that handles the overlap in case of DMA transfer

\* The overflow flag is zero

\* memmove is being called to handle the overlap

4) @brief DMA\_IRQHandler()

\* Function checks for overlap and checks if the data is being transmitted

\* If the transfer is complete, it sends Profiling completed and profiling result

PROFILING:-

1)Setup TDM   
 @brief SysTick\_Init()

\* It is used to setup TDM

\* VAL is initialized to 0

\* LOAD is set to 0xFFFFFF

\* CTRL register is set to value 5 to enable CTRL and CLK source

2) @brief Time calculation function

\* It has time\_start and time\_end function

\* pre\_interrupt uses SysTick that has a LOAD pointer

\* post\_interrupt uses SysTick that has a LOAD pointer

\* profiling time is the difference between pre and post interrupt

\* returns profiling\_time

3) @brief log\_result(uint32\_t length, uint8\_t k)

\* Function that displays the log result

4) Profiling function that is used for memset and memmove function

\* It calls standard C version

\* It calls memory function

\* It calls memmory function using Optimized O3 flag

\* It calls memset and memmove by using dma

\* Free destination and free source

PROFILING EXECUTION TIME:- in **CLOCK CYCLES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 10 Bytes | 100 Bytes | 1000 Bytes | 5000 Bytes |
| Memset(std lib) | 4 | 29 | 315 | 1457 |
| Memmove(std lib) | 4 | 37 | 404 | 1869 |
| my\_memset(O3 flag) | 4 | 31 | 341 | 1576 |
| my\_memmove(O3 flag) | 12 | 102 | 1118 | 5182 |
| Memset using dma (1byte) | 5 | 7 | 73 | 336 |
| Memmove using dma (1byte) | 8 | 10 | 77 | 343 |
| Memset using dma (2bytes) | 5 | 6 | 62 | 283 |
| Memmove using dma ( 2 bytes) | 8 | 9 | 65 | 290 |
| Memset using dma (4 bytes) | 5 | 5 | 34 | 253 |
| Memmove using dma (4 bytes) | 8 | 8 | 55 | 260 |

BINARY LOGGER:-

1)@brief log\_buffer\_configure()

\* Function that is used to configure the log\_buffer

2)

\* @brief log\_item(uint8\_t log\_id\_current)

\* Function that generates log\_item

\* It handles different cases

\* CASE Logger is initialized

\* CASE GPIO is initialized

\* CASE SYSTEM\_INITIALIZED

\* CASE SYSTEM\_HALTED

\* CASE ERROR

\* CASE INFO

\* CASE WARNING

\* CASE PROFILING\_STARTED

\* CASE PROFILING\_RESULT

\* CASE DATA\_RECEIVED

\* CASE PROFILING\_COMPLETED

\* CASE DATA\_ANALAYSIS\_STARTED

\* CASE DATA\_ALPHA\_COUNT

\* CASE DATA\_PUNCTUATION\_COUNT

\* CASE DATA\_NUMERIC\_COUNT

\* CASE DATA\_MISC\_COUNT

\* CASE DATA\_ANALYSIS\_COMPLETED

\* CASE HEARTBEAT

3) @brief log\_data

\* Function that pushes the log data

\* returns the count for data

4) @brief log\_string

\* Function that passes a log string

\* returns the count for string

5)

@brief log\_integer

\* Function that passes a log integer

\* returns the count for integer values

6) @brief log\_flush

\* Function that flushes out the log

\* returns success or failure