Project 2 Report Template

Date-7th Nov 2018

Your Name –Saloni Shambhuwani

Unity ID-sdshambh

Student ID-200266197

Email Address-sdshambh@ncsu.edu

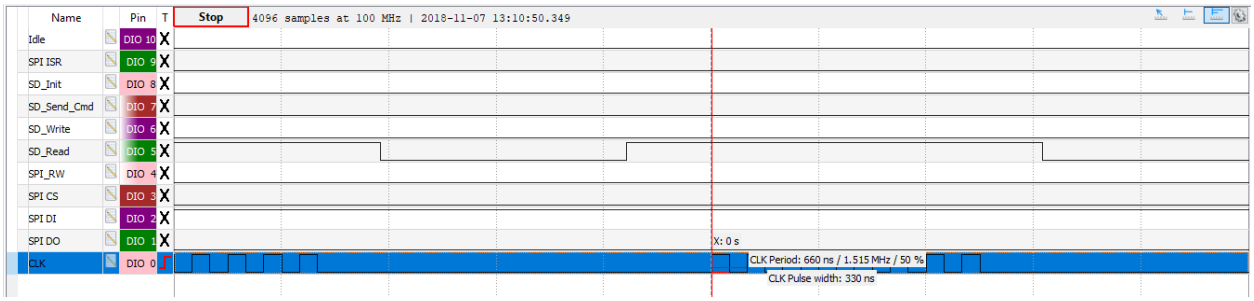
Note: For the requested diagrams, scans or photos of hand-drawn diagrams are sufficient for full credit.

Part A

Timing Analysis

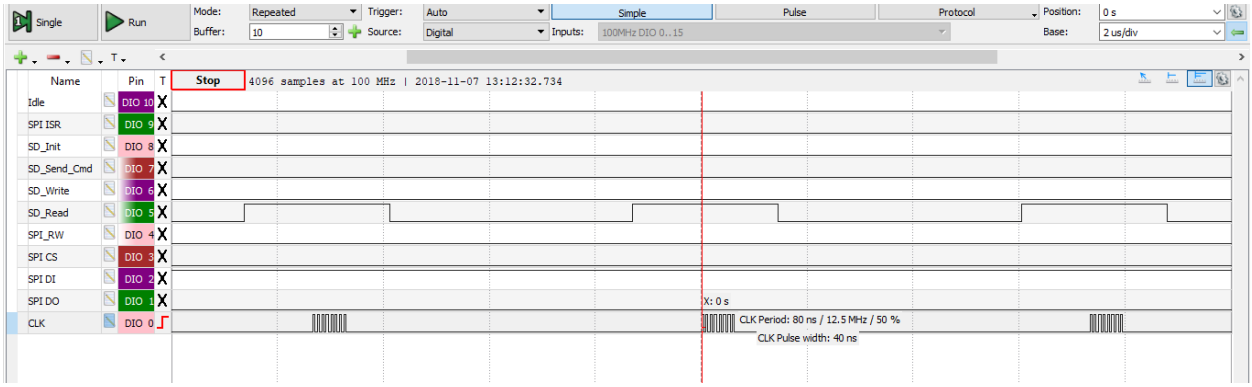
1. What is the original SPI bit rate?

Ans:1.515MHz



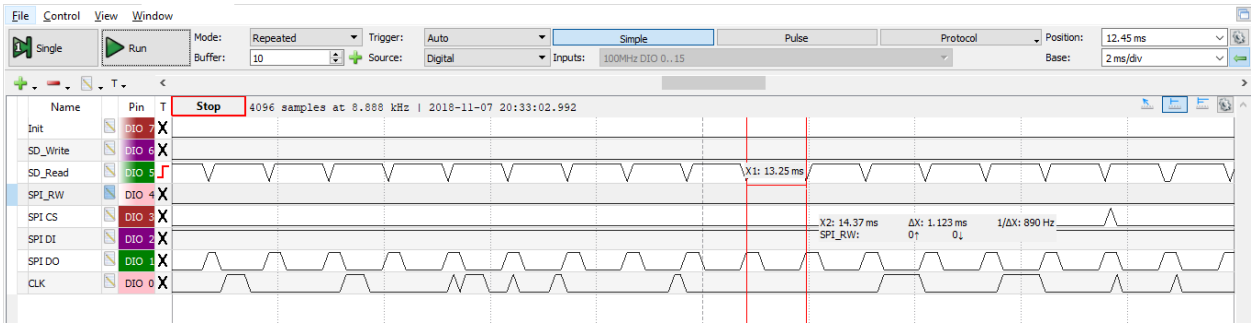
2. What is the fastest SPI bit rate which works for your µSD card?

Ans: 12.5MHz

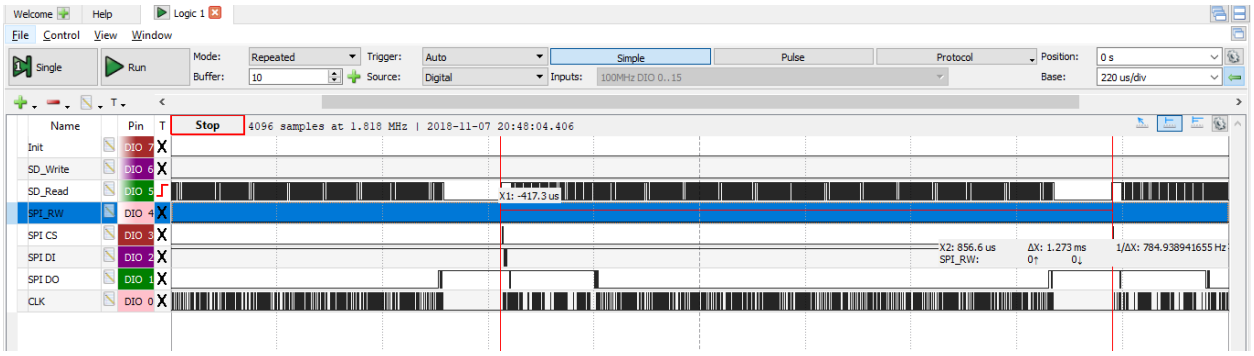


5. **Logic analyzer screen shot showing SPI and debug signals for during SD_Read or SD_Write operation (*not* SD_Init), marked with segments (Compute, I/O-SD, I/O-SPI, Other).**

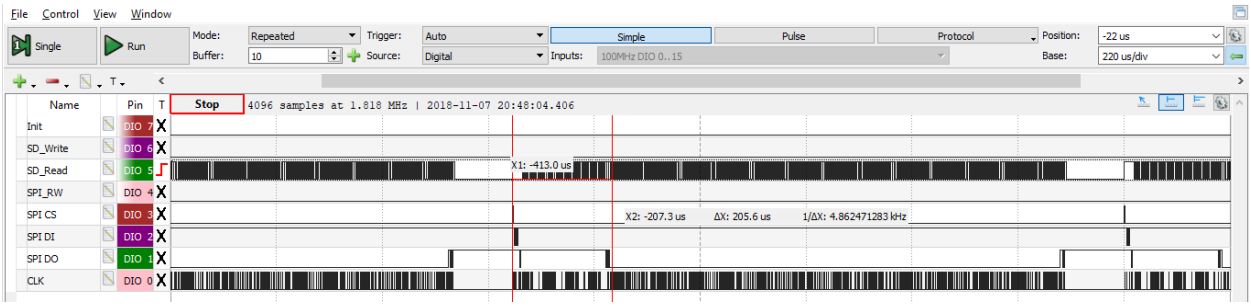
Ans: SD_READ



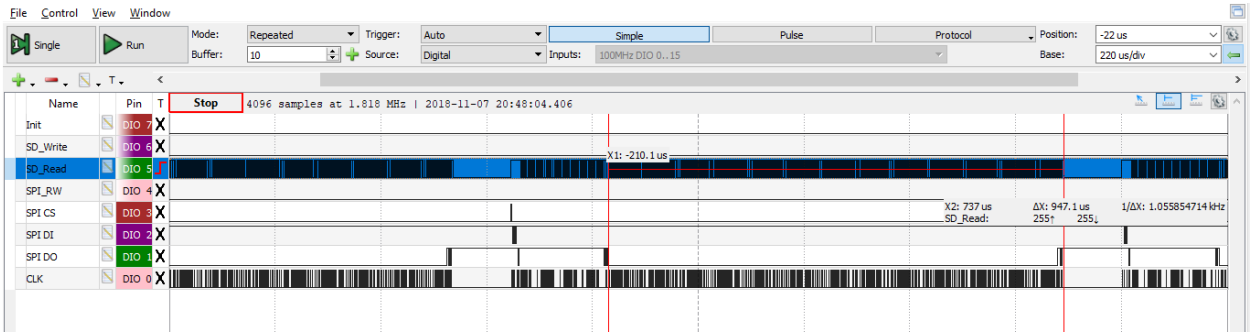
Total Read time: 1.273ms



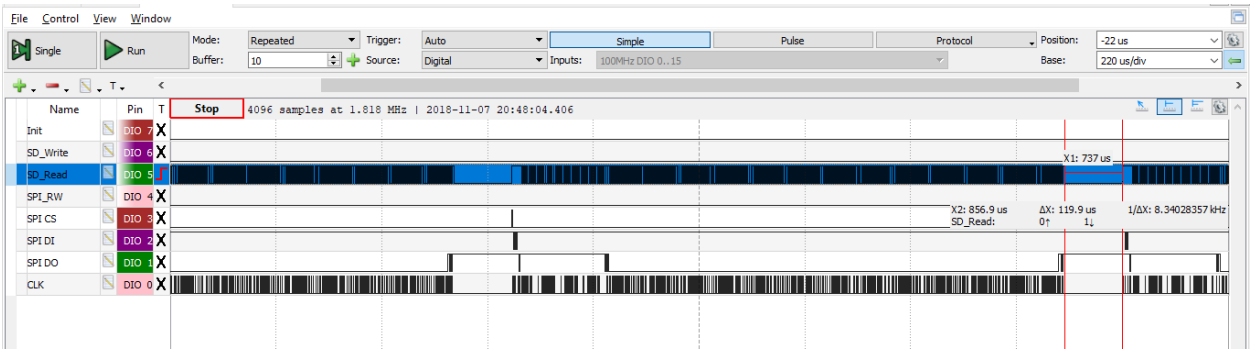
SD time:205us



SPI time:947 us

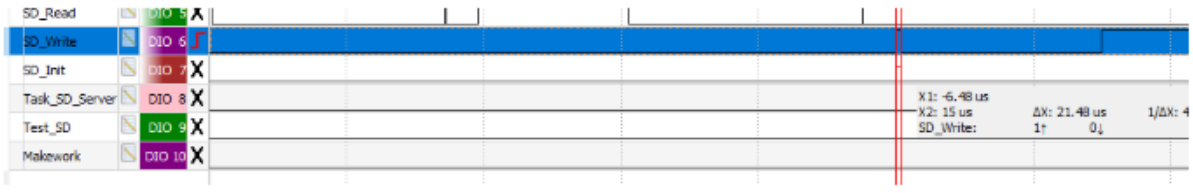


Others:129.9us

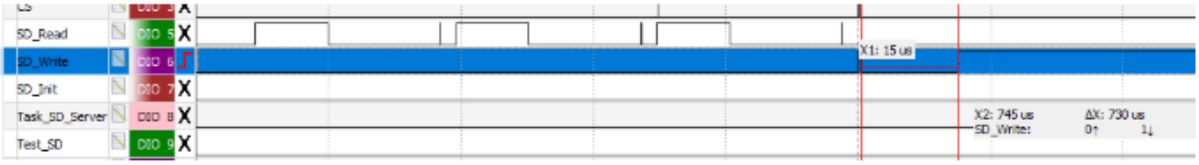


Write Time:

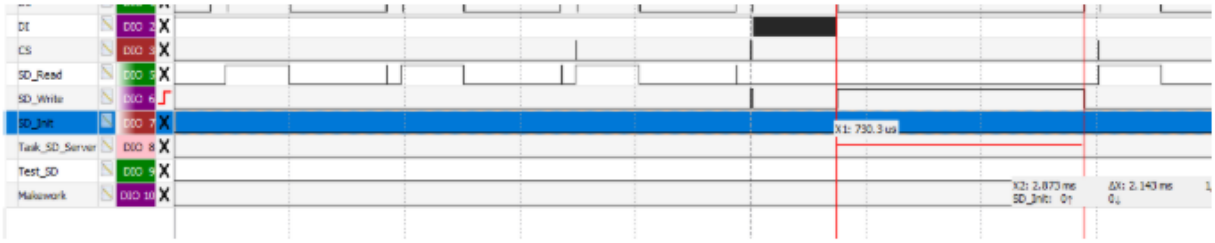
Computation time:21.48us



Computation and SD time: 745us



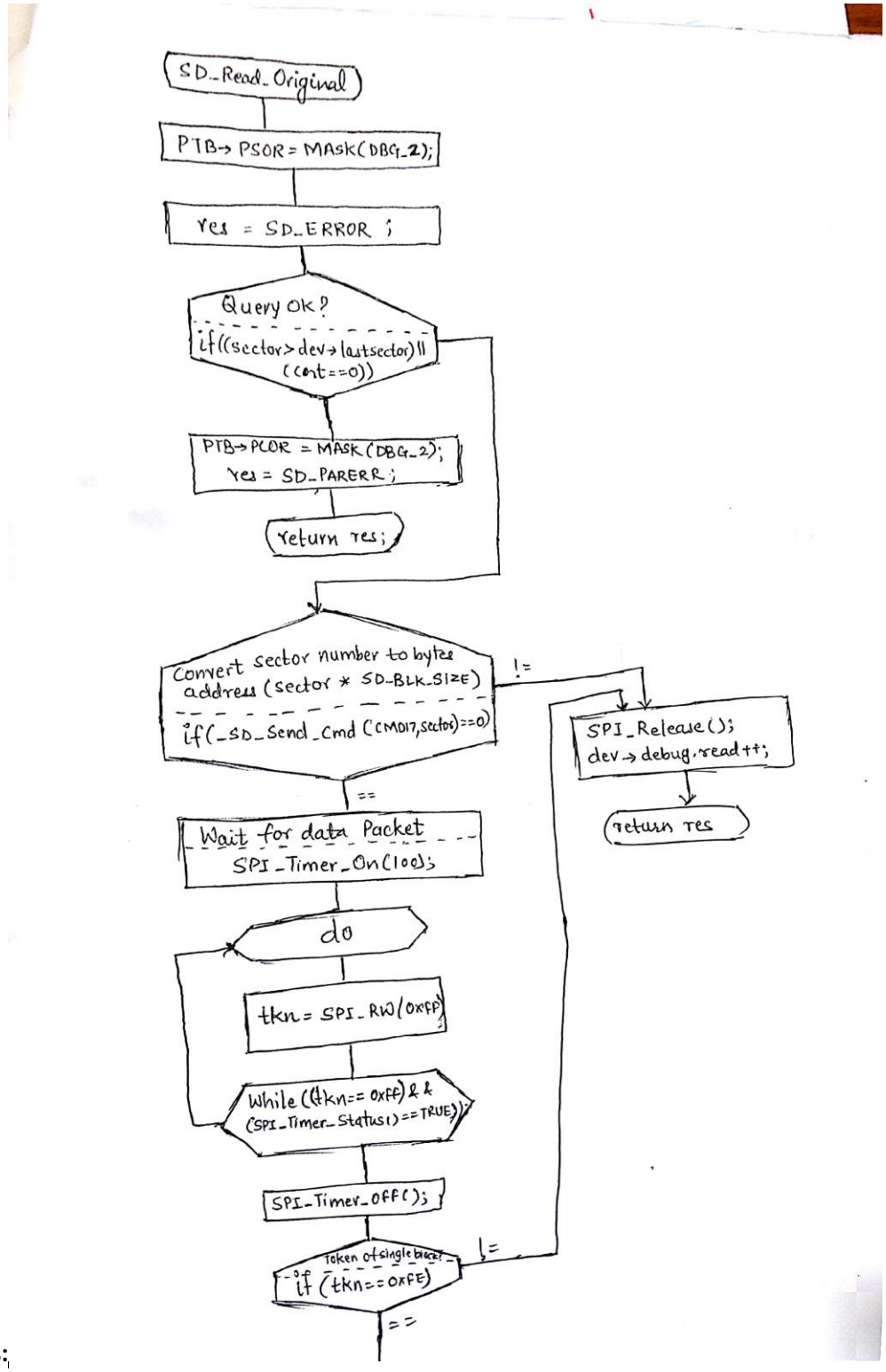
SPI time and computation:2.875ms

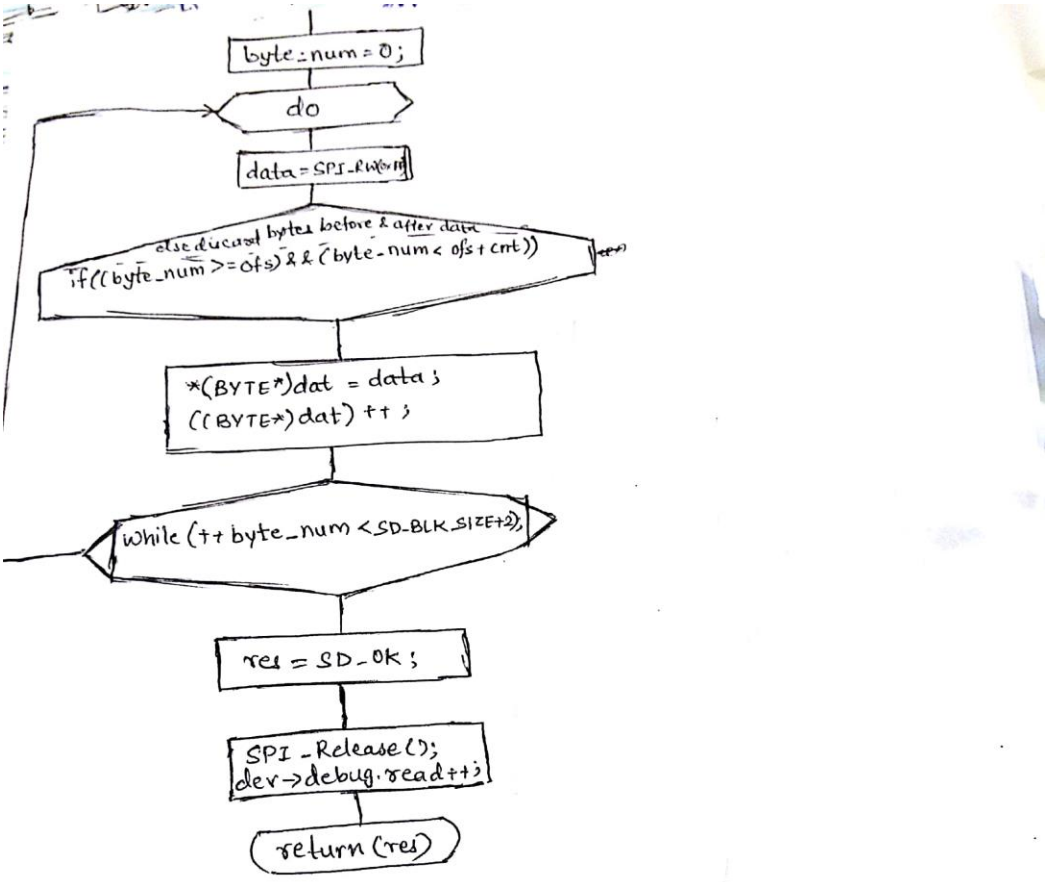


Code Structure Analysis

6. **Control flow graph for SD_Read.**

Ans:

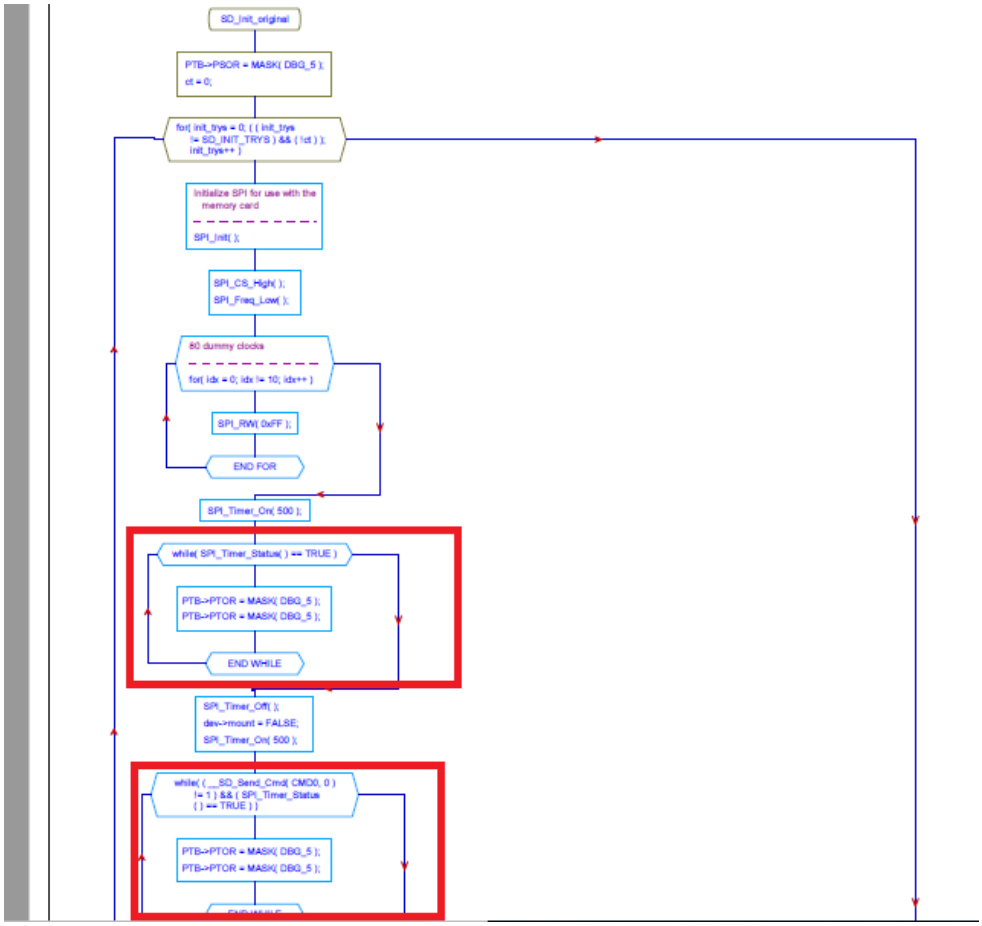


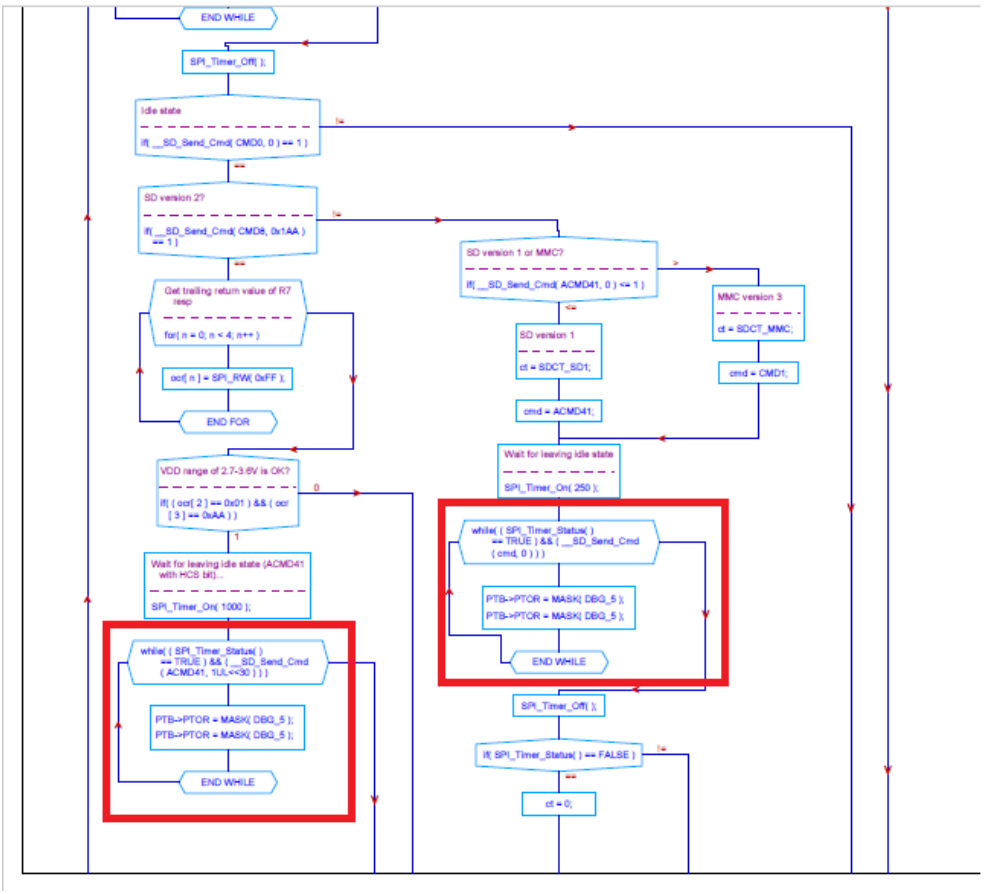


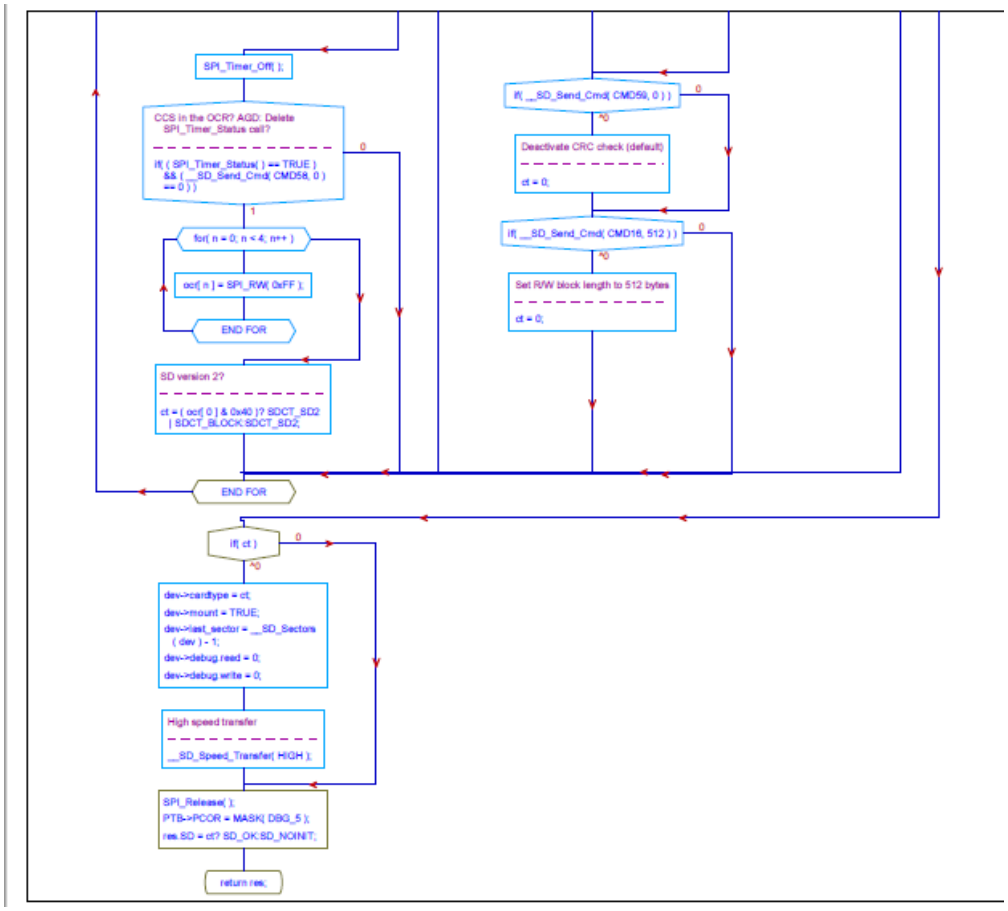
7. **Control flow graphs highlighting operations which may block (i.e. repeat a loop an unknown time number of times).**

Ans: Highlighted code may block.

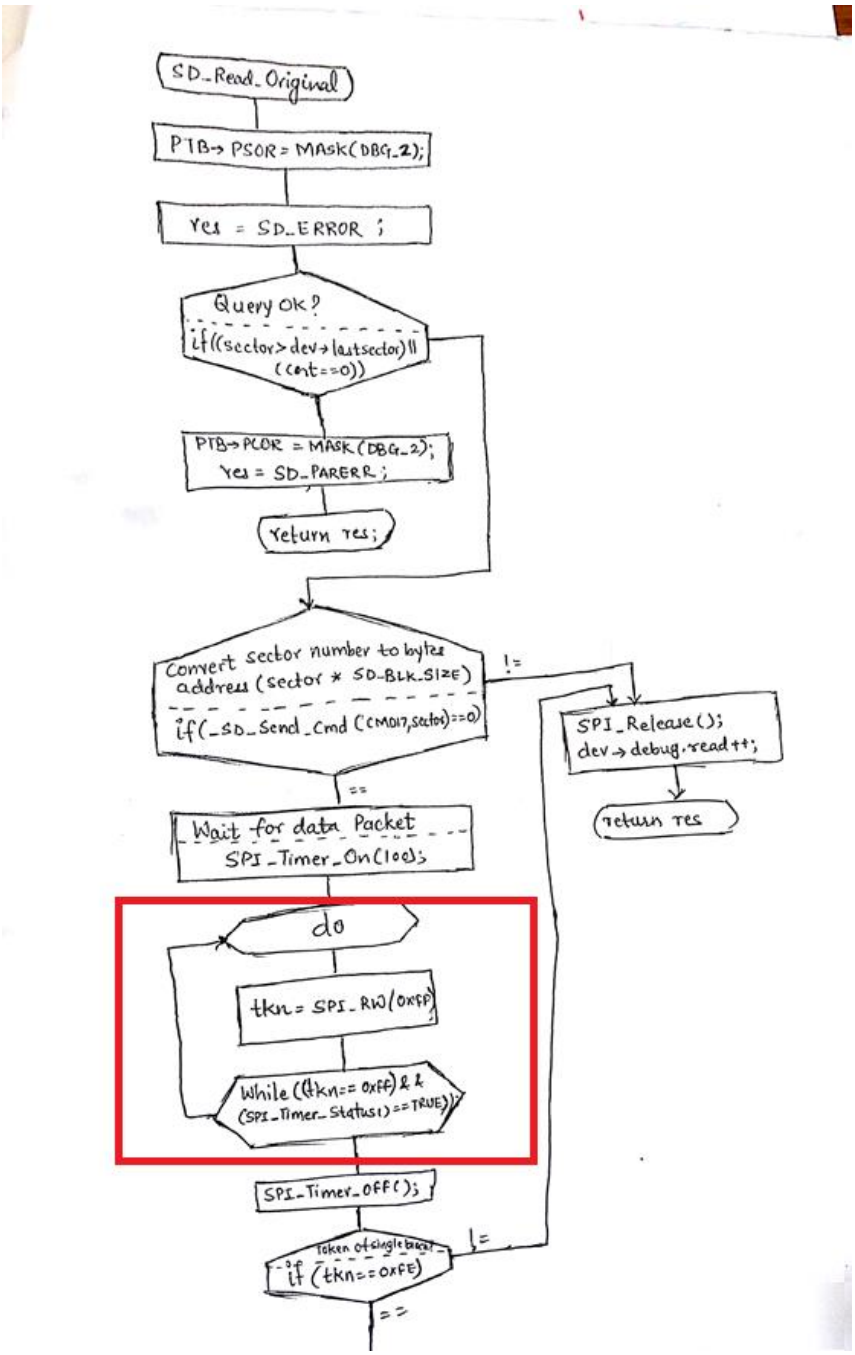
   a. **SD_Init**

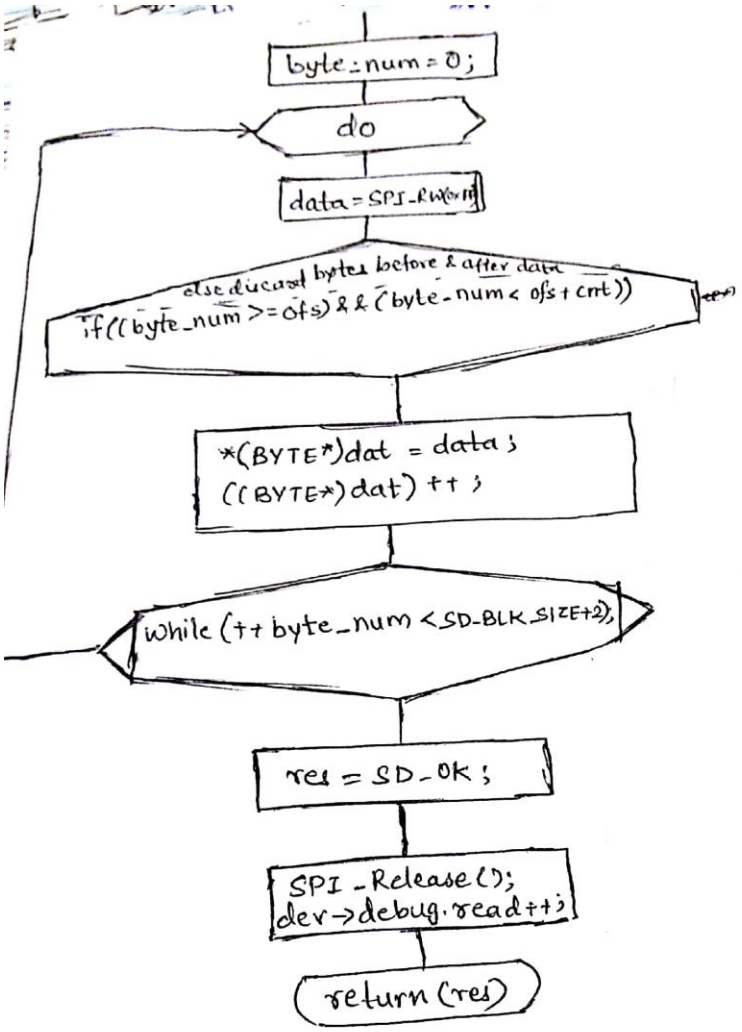

SD_Init_original
PTB->PSOR = MASK(DBG_5);
ct = 0;
for(init_trys = 0; ((init_trys != SD_INIT_TRYS) && (!ct)); init_trys++)
Initialize SPI for use with the memory card
SPI_Init( );
SPI_CS_High( );
SPI_Freq_Low( );
80 dummy clocks
for(idx = 0; idx != 10; idx++)
SPI_RW(0xFF);
END FOR
SPI_Timer_On(500);
while(SPI_Timer_Status( ) == TRUE)
PTB->PTOR = MASK(DBG_5);
PTB->PTOR = MASK(DBG_5);
END WHILE
SPI_Timer_Off( );
dev->mount = FALSE;
SPI_Timer_On(500);
while((__SD_Send_Cmd(CMD0, 0) != 1) && (SPI_Timer_Status ( ) == TRUE))
PTB->PTOR = MASK(DBG_5);
PTB->PTOR = MASK(DBG_5);


END WHILE
SPI_Timer_Off( );
Idle state
if(__SD_Send_Cmd(CMD0, 0) == 1)
!=
==
SD version 2?
if(__SD_Send_Cmd(CMD8, 0x1AA) == 1)
!=
==
Get trailing return value of R7 resp
for(n = 0; n < 4; n++)
ocr[n] = SPI_RW(0xFF);
END FOR
VDD range of 2.7-3.6V is OK?
if((ocr[2] == 0x01) && (ocr [3] == 0xAA))
0
1
Wait for leaving idle state (ACMD41 with HCS bit)...
SPI_Timer_On(1000);
while((SPI_Timer_Status( ) == TRUE) && (__SD_Send_Cmd (ACMD41, 1UL<<30)))
PTB->PTOR = MASK(DBG_5);
PTB->PTOR = MASK(DBG_5);
END WHILE
SD version 1 or MMC?
if(__SD_Send_Cmd(ACMD41, 0) <= 1)
>
<=
MMC version 3
ct = SDCT_MMC;
cmd = CMD1;
SD version 1
ct = SDCT_SD1;
cmd = ACMD41;
Wait for leaving idle state
SPI_Timer_On(250);
while((SPI_Timer_Status( ) == TRUE) && (__SD_Send_Cmd (cmd, 0)))
PTB->PTOR = MASK(DBG_5);
PTB->PTOR = MASK(DBG_5);
END WHILE
SPI_Timer_Off( );
if(SPI_Timer_Status( ) == FALSE)
!=
==
ct = 0;

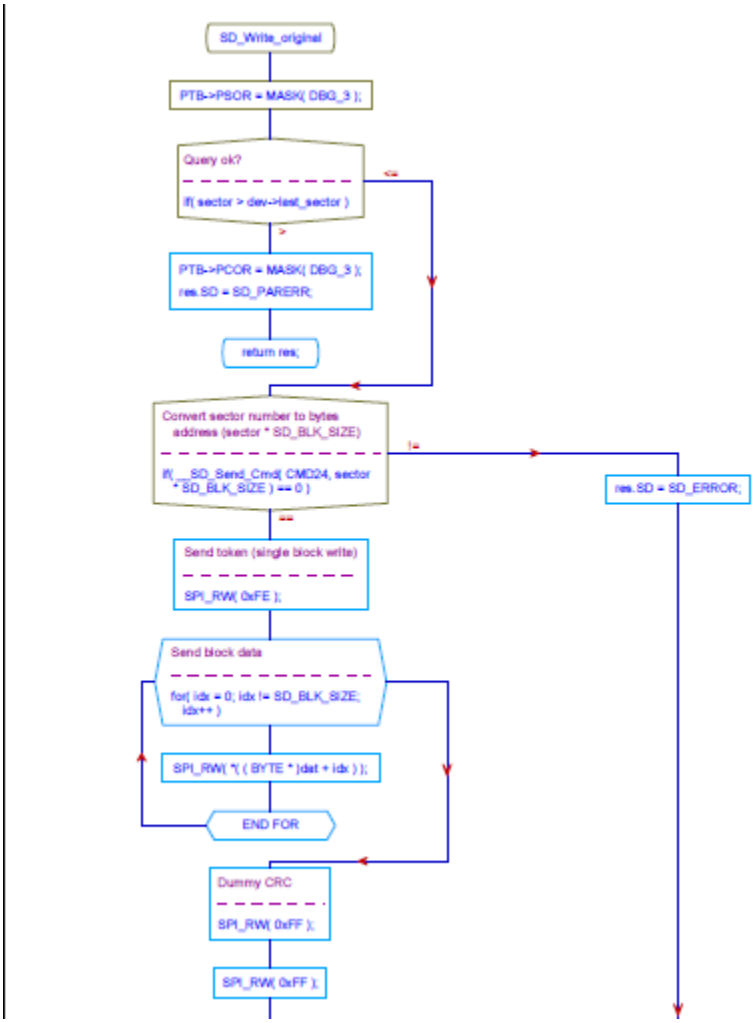

SPI_Timer_Off( );
CCS in the OCR? AGD: Delete SPI_Timer_Status call?
if((SPI_Timer_Status( ) == TRUE) && (__SD_Send_Cmd(CMD58, 0) == 0))
0
1
for(n = 0; n < 4; n++)
ocr[n] = SPI_RW(0xFF);
END FOR
SD version 2?
ct = (ocr[0] & 0x40)? SDCT_SD2 | SDCT_BLOCK:SDCT_SD2;
END FOR
if(__SD_Send_Cmd(CMD59, 0))
0
*0
Deactivate CRC check (default)
ct = 0;
if(__SD_Send_Cmd(CMD16, 512))
0
*0
Set R/W block length to 512 bytes
ct = 0;
if(ct)
0
*0
dev->cardtype = ct;
dev->mount = TRUE;
dev->last_sector = __SD_Sectors (dev) - 1;
dev->debug.read = 0;
dev->debug.write = 0;
High speed transfer
__SD_Speed_Transfer(HIGH);
SPI_Release( );
PTB->PCOR = MASK(DBG_5);
res.SD = ct? SD_OK:SD_NOINIT;
return res;

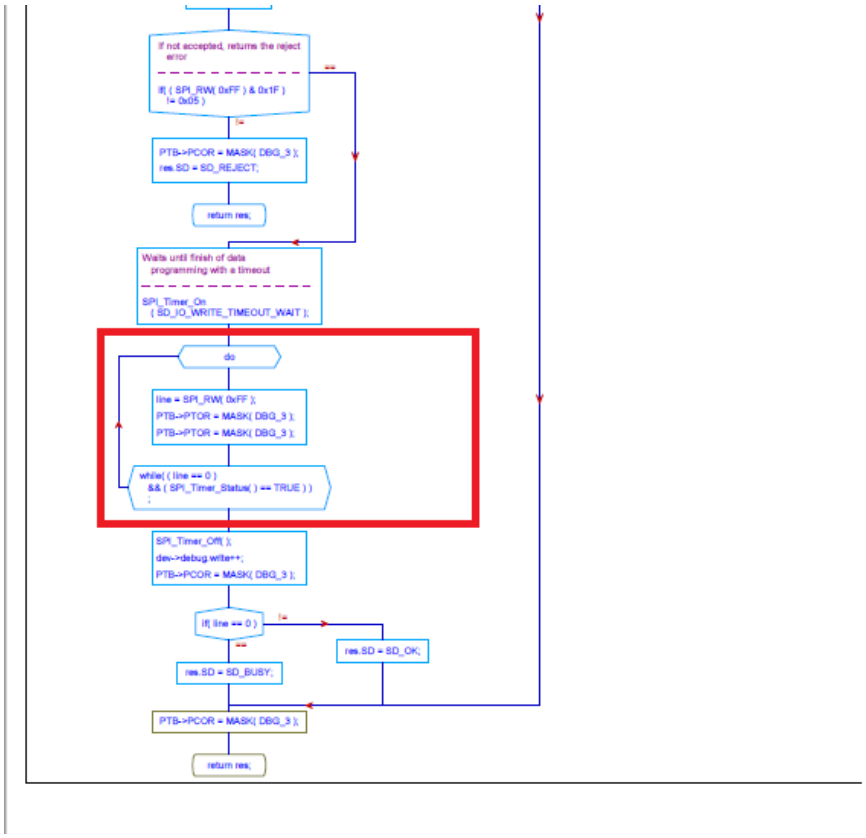
b. SD_Read


SD_Read_Original
PTB→ PSOR = MASK(DBG_2);
Yes = SD_ERROR ;
Query OK?
if((sector> dev→ lastsector) ||
((cnt==0))
PTB→ PCOR = MASK(DBG_2);
Yes = SD_PARERR ;
return res;
Convert sector number to bytes address (sector * SD_BLK_SIZE)
if(_SD_Send_Cmd (CMD17,sector)==0)
!=
SPI_Release();
dev→ debug.read++;
return res
==
Wait for data Packet
SPI_Timer_On(100);
do
tkn = SPI_RW(0xFF)
While ((tkn== 0xFF) && (SPI_Timer_Status1) == TRUE))
SPI_Timer_OFF();
Token of single block
if (tkn==0xFE)
!=
==


byte_num = 0;
do
data = SPI_RW(0xff);
else discard bytes before & after data
if((byte_num >= ofs) && (byte_num < ofs + cnt))
(BYTE)dat = data;
((BYTE*)dat)++;
while (++byte_num < SD_BLK_SIZE+2);
res = SD_OK;
SPI_Release();
dev->debug.read++;
return (res)

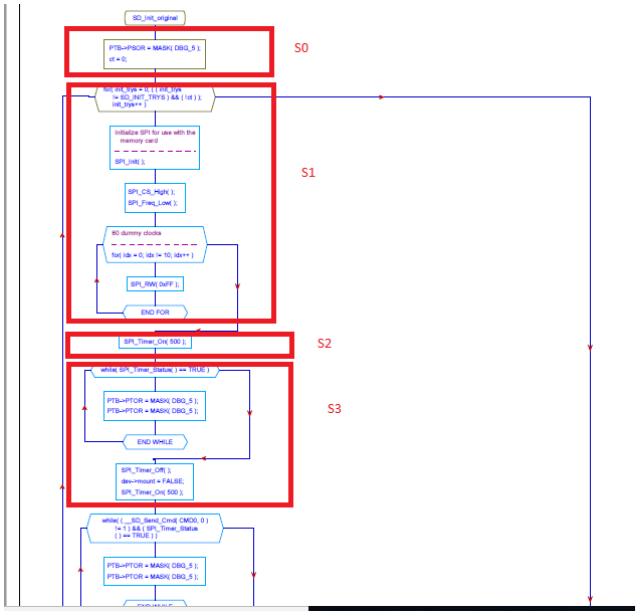
c. SD_Write

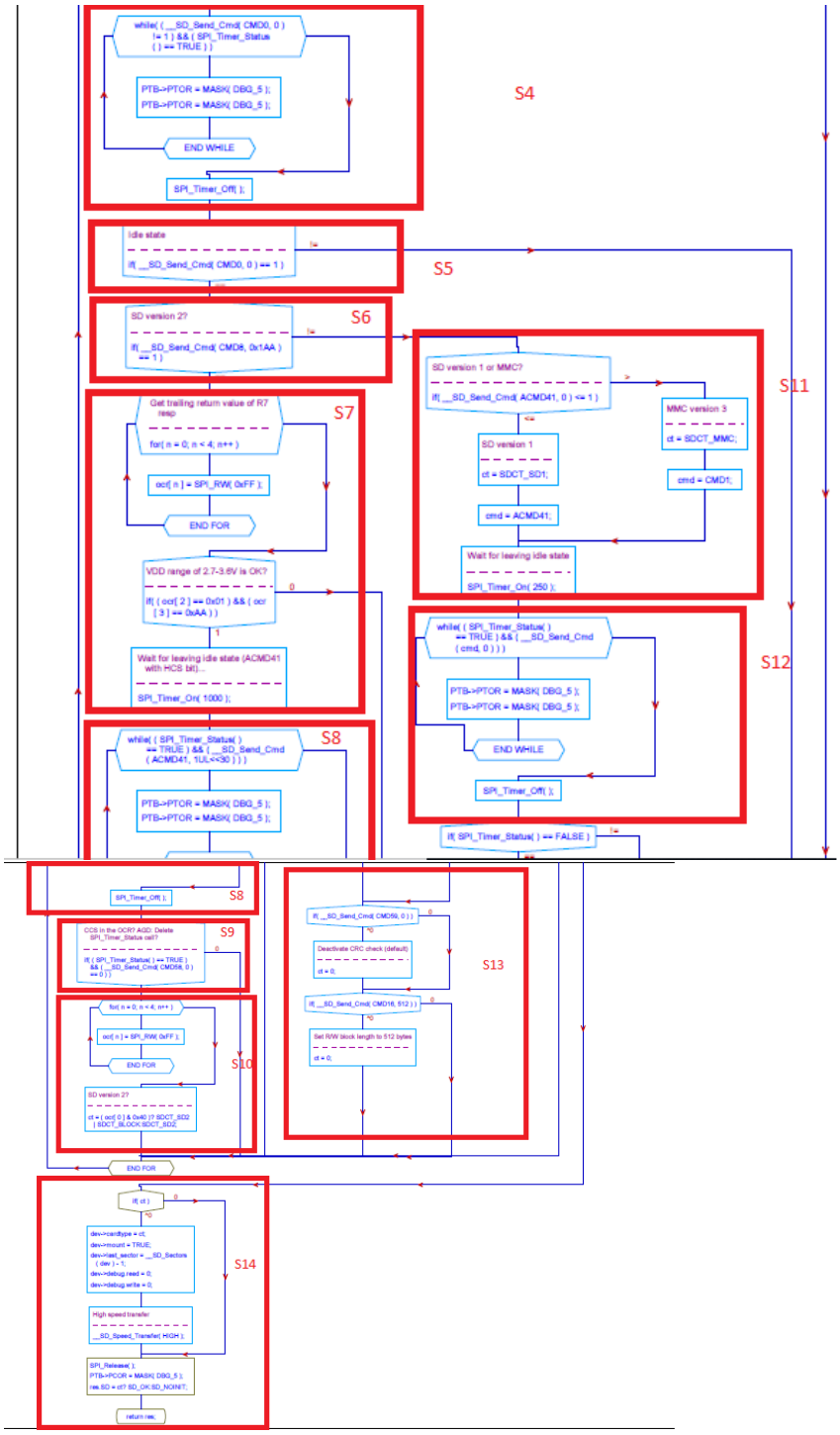

SD_Write_original
PTB->PSOR = MASK(DBG_3);
Query ok?
if(sector > dev->last_sector)
<=
>
PTB->PCOR = MASK(DBG_3);
res.SD = SD_PARERR;
return res;
Convert sector number to bytes address (sector * SD_BLK_SIZE)
if(__SD_Send_Cmd(CMD24, sector * SD_BLK_SIZE) == 0)
!=
res.SD = SD_ERROR;
==
Send token (single block write)
SPI_RW(0xFE);
Send block data
for(idx = 0; idx != SD_BLK_SIZE; idx++)
SPI_RW(*((BYTE *)dat + idx));
END FOR
Dummy CRC
SPI_RW(0xFF);
SPI_RW(0xFF);



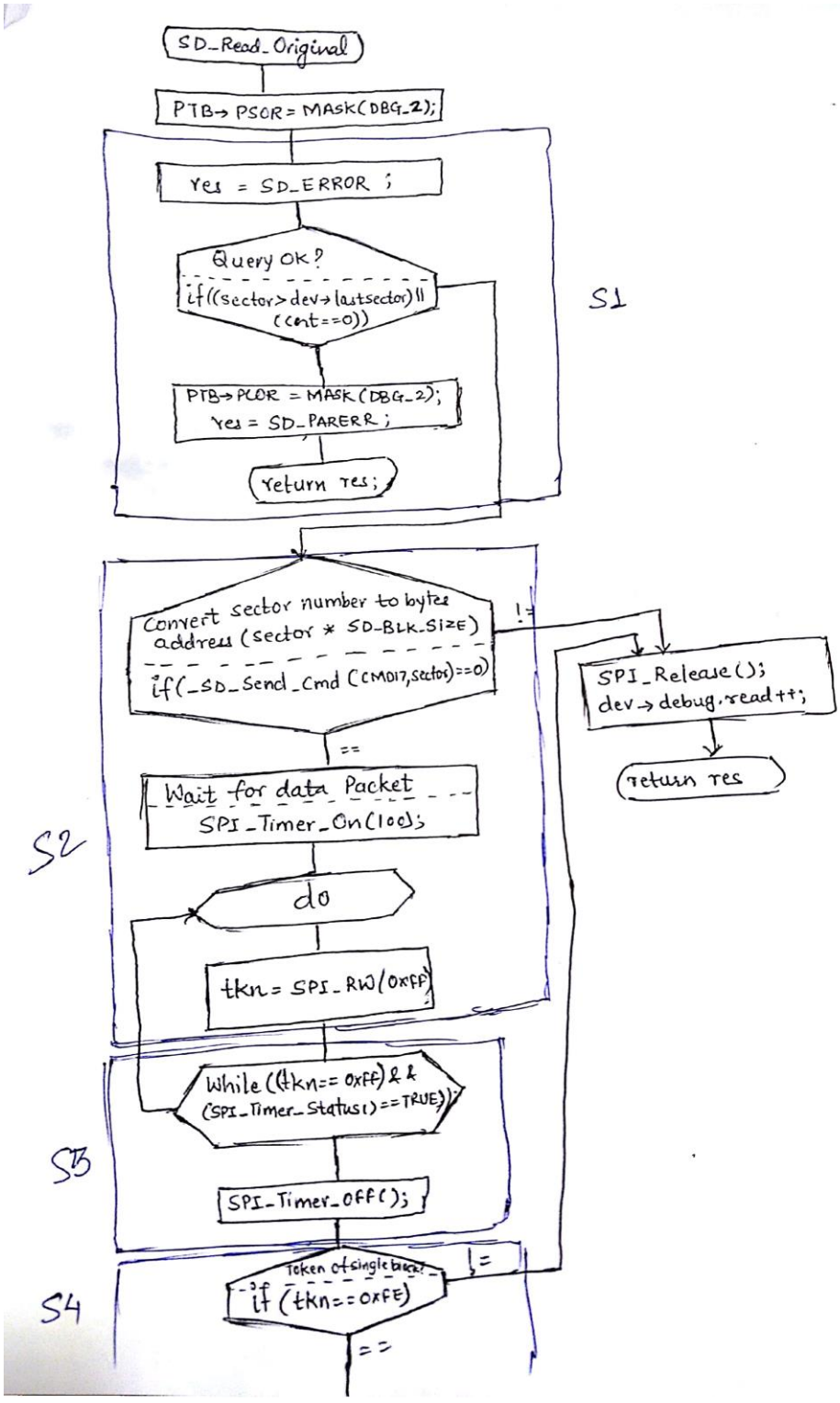
Code Transformation

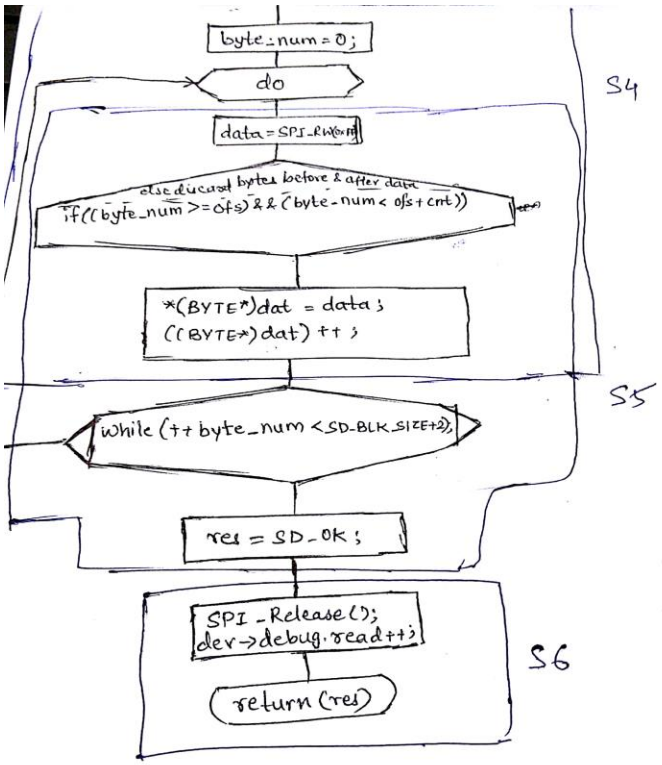
8. **CFG with FSM states overlaid**
    a. **Marked-up control flow graphs allocating code to states**
        i. **SD_Init**



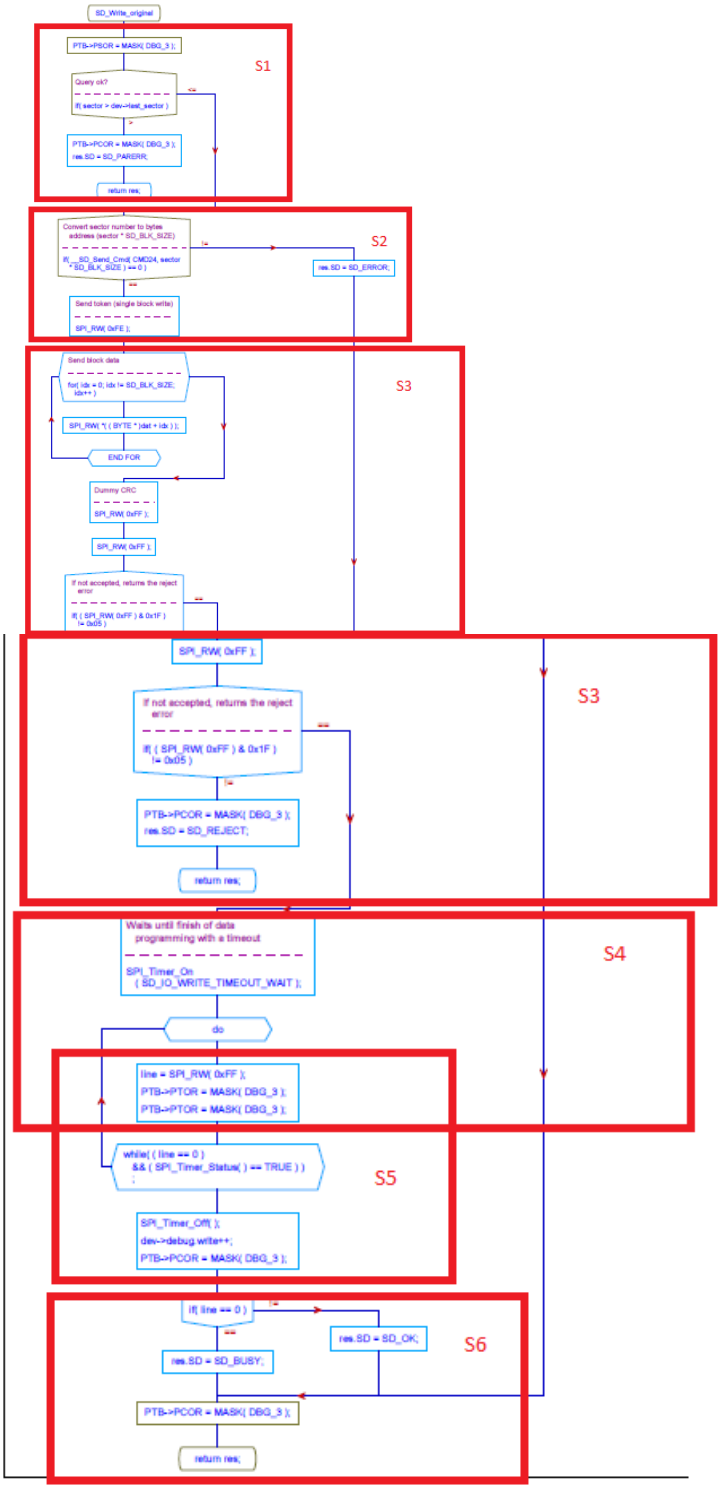

S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
END WHILE
SPI_Timer_Off( );
END FOR
SPI_Timer_On(1000);
SPI_Timer_On(250);
PTB->PTOR = MASK(DBG_5);
PTB->PTOR = MASK(DBG_5);

ii. SD_Read


SD_Read_Original
PTB→ PSOR = MASK(DBG_2);
Yes = SD_ERROR ;
Query OK ?
if((sector>dev→lastsector)||(cnt==0))
PTB→PCOR = MASK(DBG_2);
Yes = SD_PARERR ;
return res;
S1
Convert sector number to bytes address (sector * SD_BLK_SIZE)
if(_SD_Send_Cmd (CMD17,sector)==0)
!=
==
SPI_Release();
dev→debug.read++;
return res
Wait for data Packet
SPI_Timer_On(100);
S2
do
tkn = SPI_RW(0xFF);
While ((tkn== 0xFF) && (SPI_Timer_Status)==TRUE));
S3
SPI_Timer_Off();
Token of single block?
if (tkn==0xFE)
!=
==
S4


byte_num = 0;
do
S4
data = SPI_RW(0xff);
else discard bytes before & after data
if((byte_num >= ofs) && (byte_num < ofs+cnt))
(BYTE)dat = data;
(BYTE*)dat)++;
S5
while (++byte_num < SD_BLK_SIZE+2);
res = SD_OK;
SPI_Release();
dev->debug.read++;
return (res)
S6

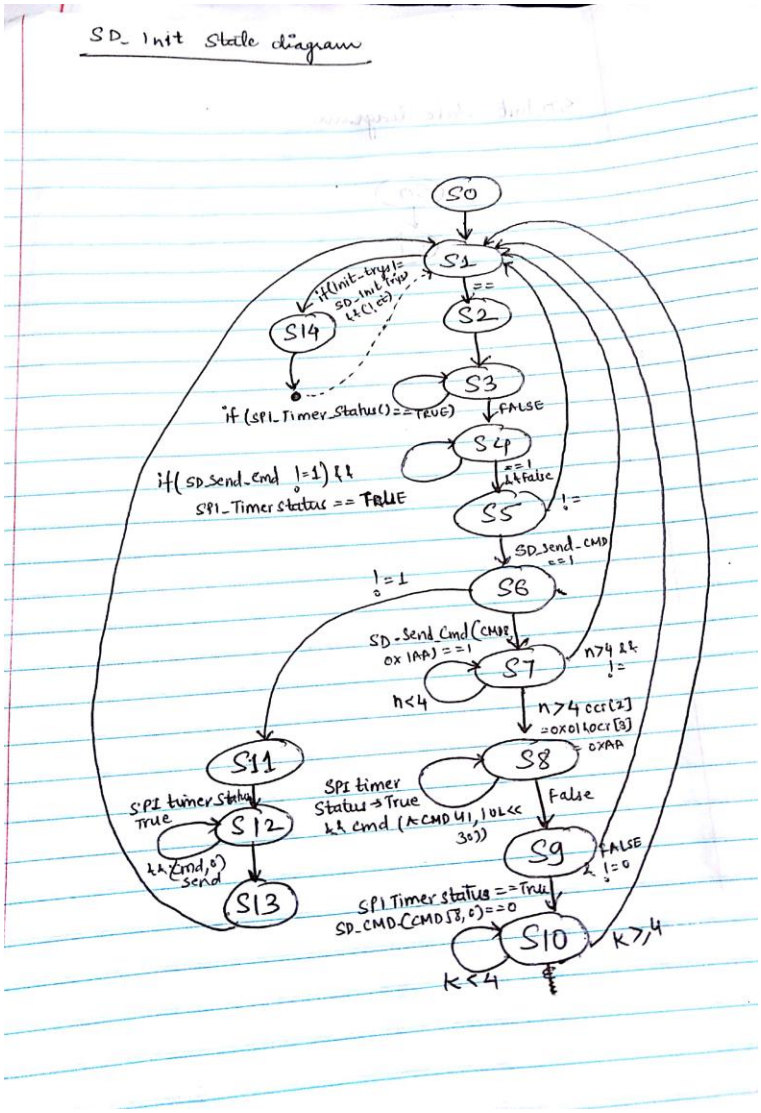
iii. SD_Write



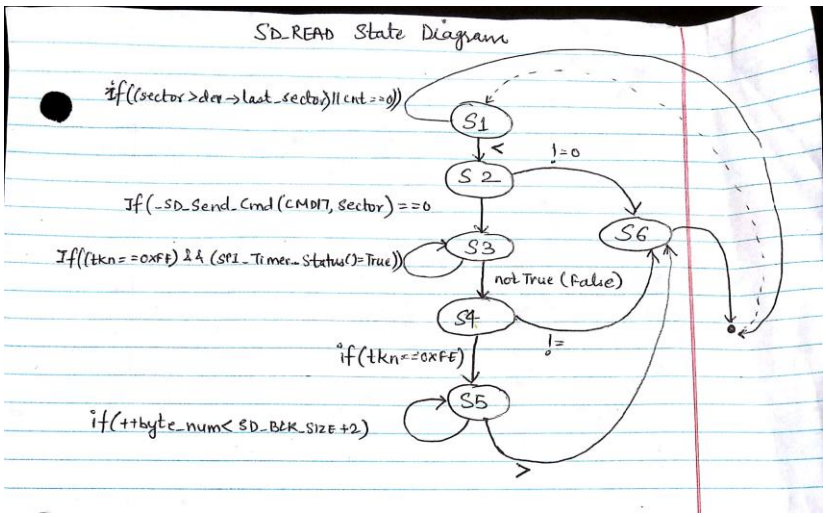
9. FSM diagrams

a. State diagram with both states and transitions labeled

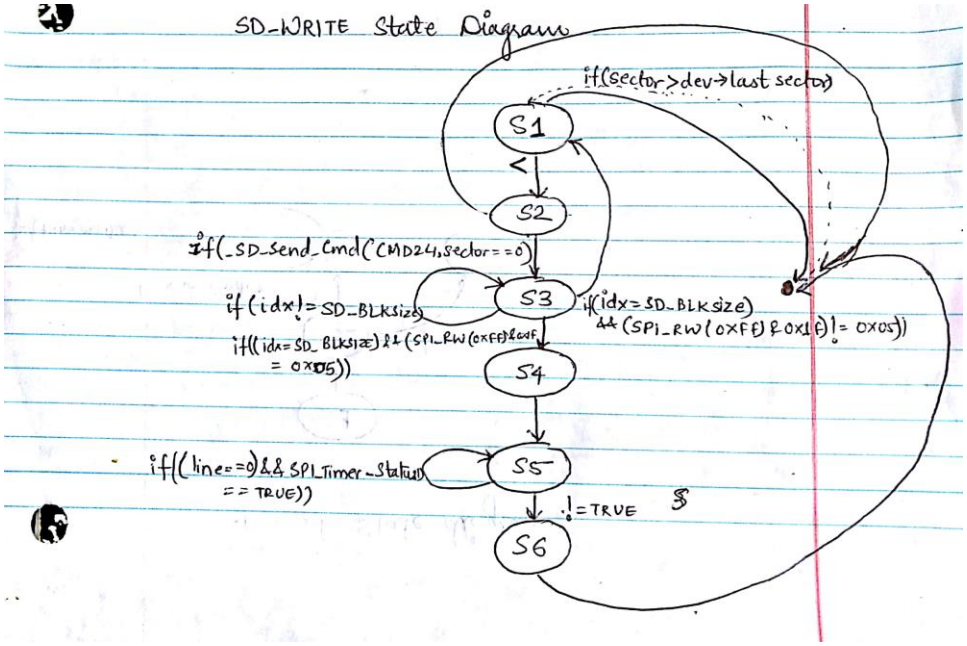
i. **SD_Init**



ii. **SD_Read**



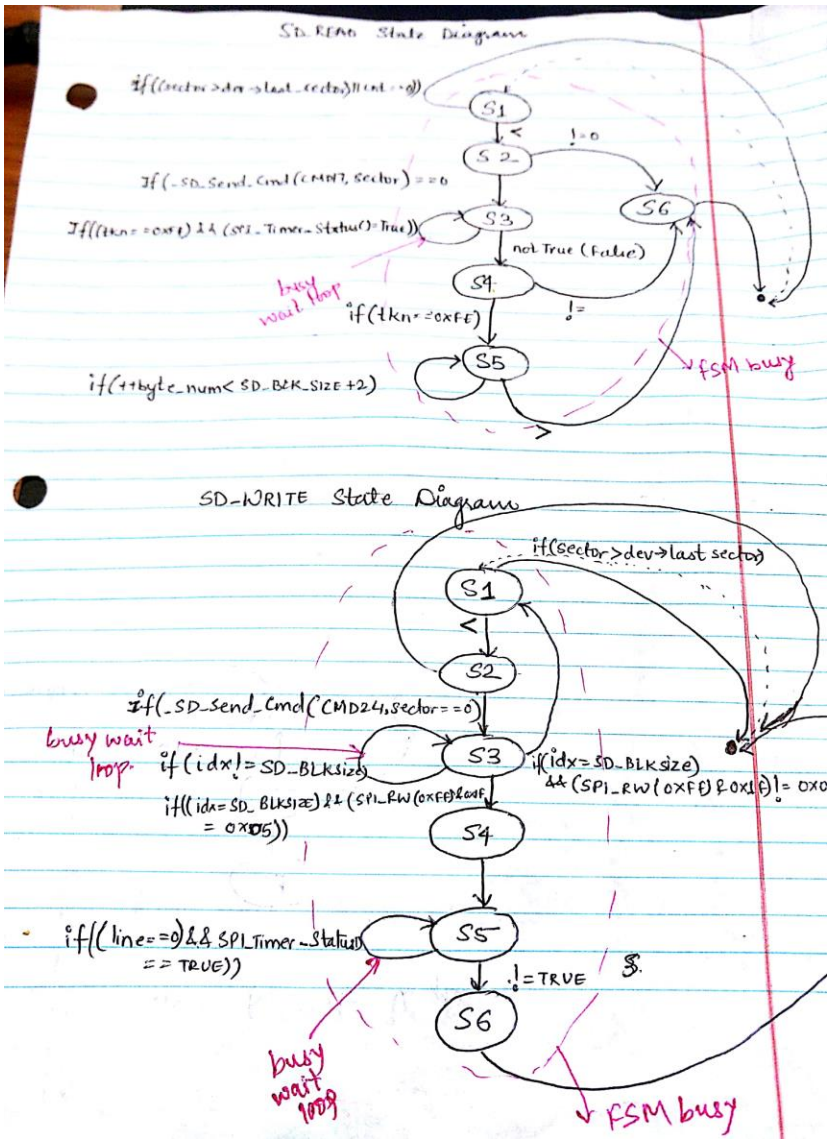
iii. **SD_Write**



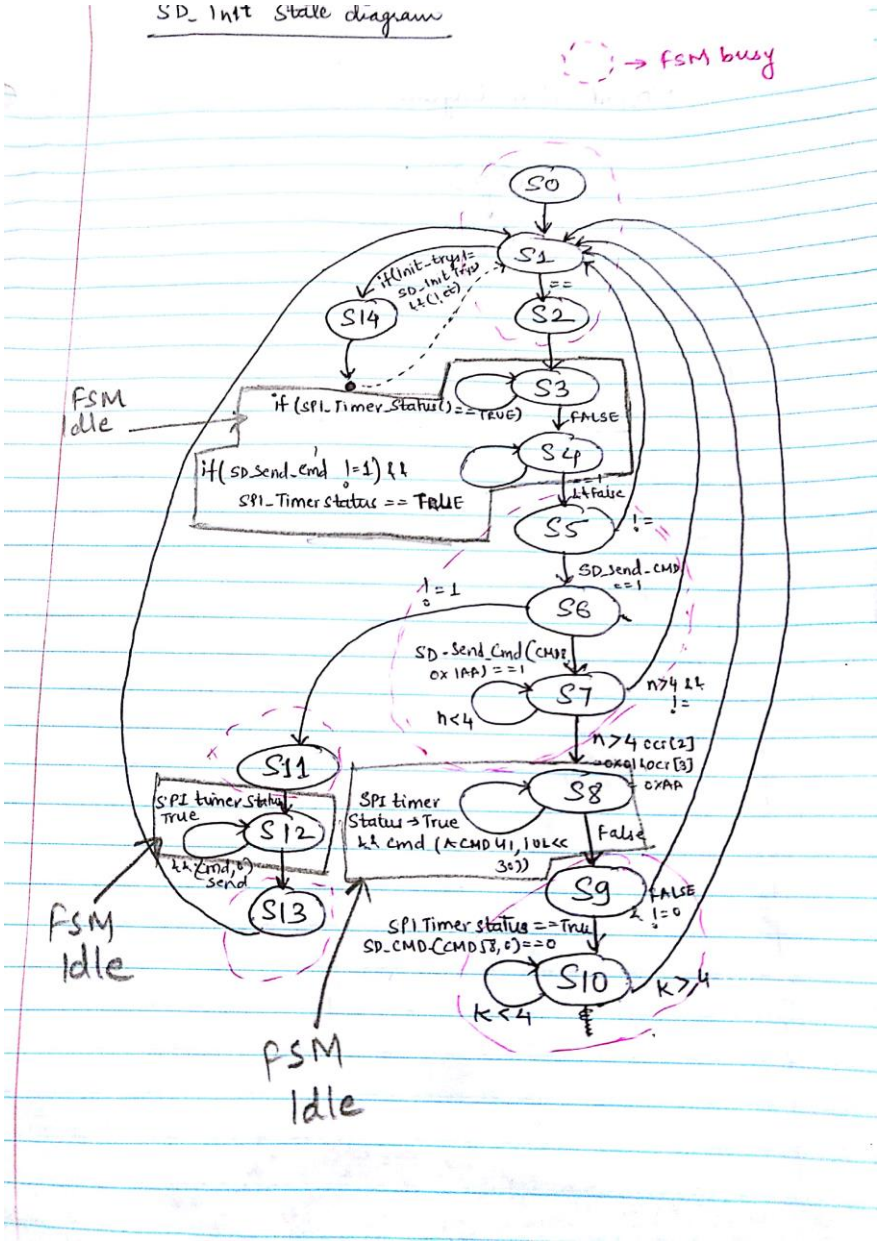
b. State diagram showing which states are busy and which are idle.

i. **SD_Read**

ii. **SD_Write**


SD READ State Diagram
S1
S2
S3
S4
S5
S6
busy wait loop
FSM busy
SD_WRITE State Diagram
if(sector>dev->last sector)
busy wait loop.
busy wait loop
FSM busy

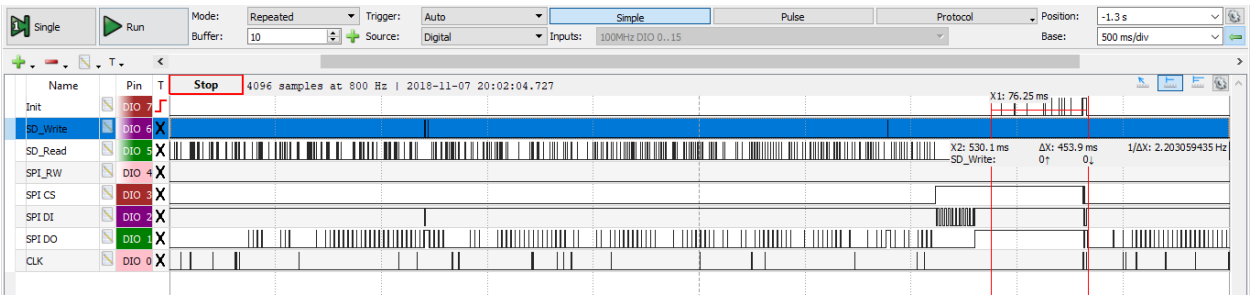
i. **SD_Init**



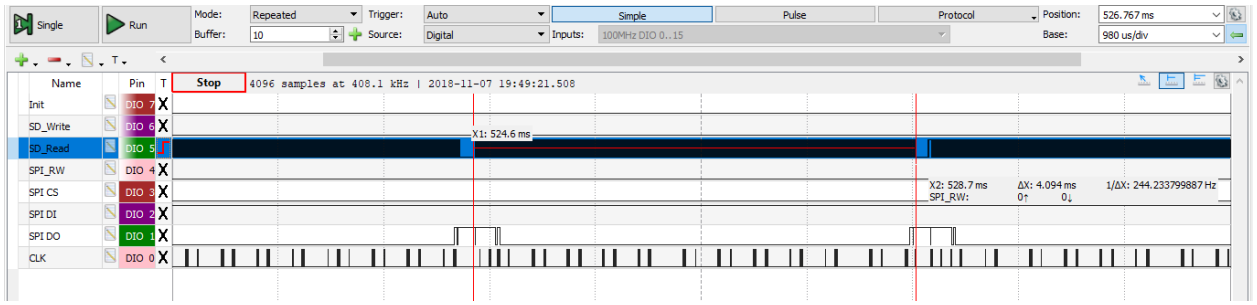
10. FSM Verification

a. Logic analyzer screen shot showing SPI signals (SPI CLK, SPI DI, SPI DO, SPI CS) and debug signals (SD_Read, SD_Write, SD_Init, test_write).

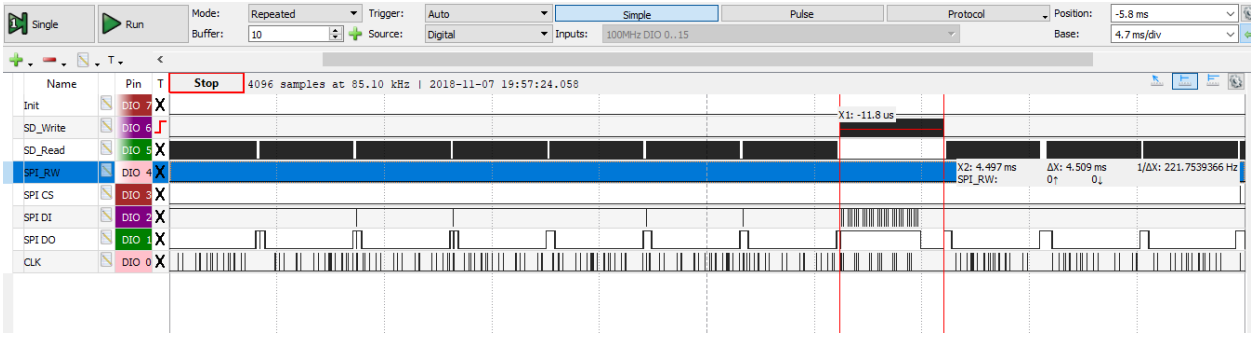
i. SD_Init operation: Total operation time for SD_Init is 453.9ms



ii. SD_Read operation: total read operation time with FSM is 4.054ms

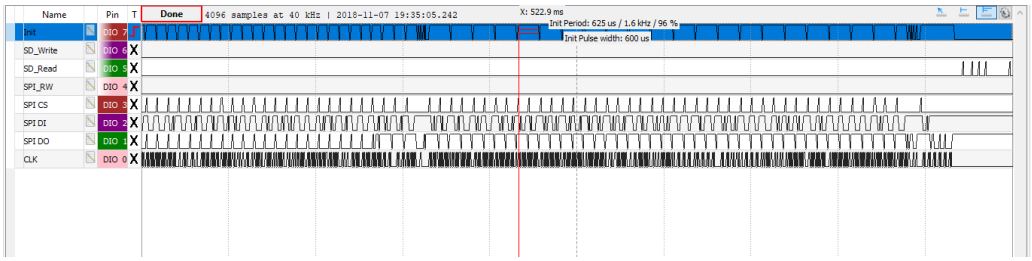


iii. **SD_Write operation: 4.509ms is the total write operation time.**

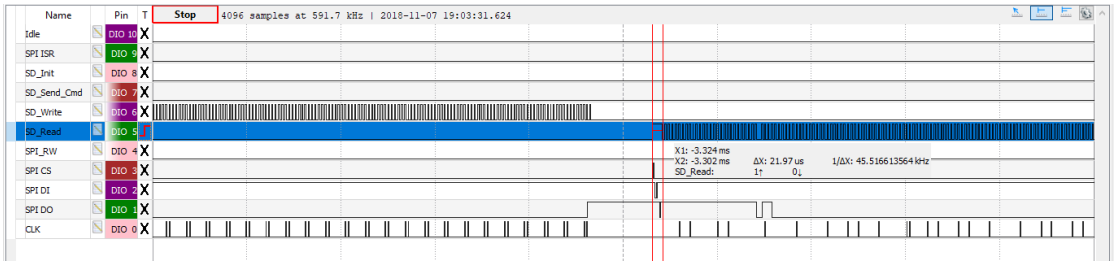


b. **Find and analyze the state with longest code. List the state name and the maximum execution time observed. Note: this is the state which takes the longest time to execute its code once. It is not the total time spent in the state, or the total time spent executing this state's code.**

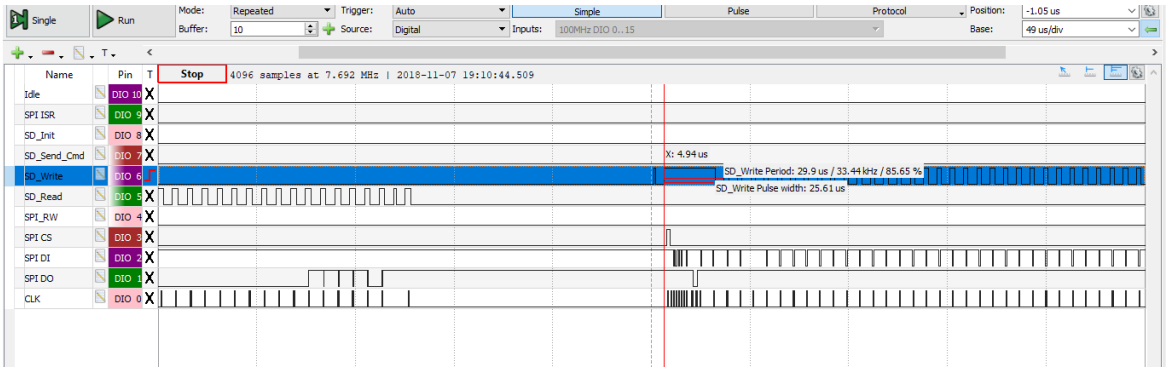
i. **SD_Init operation**



ii. **SD_Read operation**



iii. **SD_Write operation**



c. Table with maximum state execution times of your FSM-based code.

Function	Name and Duration of State with Longest Code
SD_Init	Case:S8; 600 **µs**
SD_Read	Case:S1; 21.97 **µs**
SD_Write	Case S2; 25.61 **µs**