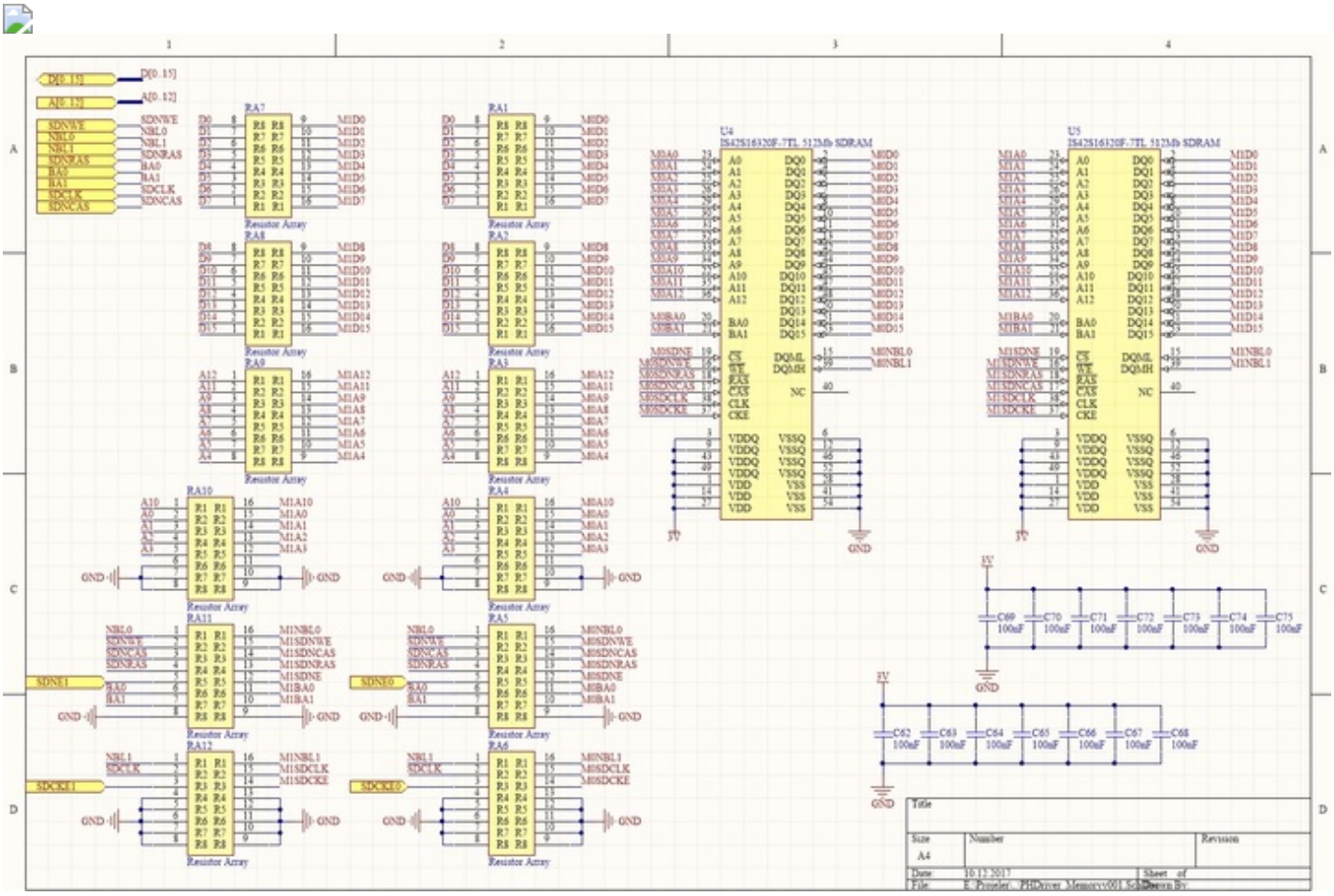
Hello, I've designed a board with the MCU and SDRAM that are specified in the header. But I've a problem about the read and write. The data in the memory on the Keil is changing independently from my code. I think this is a refresh or some timing issue but I am not able to find it. This is my first experience with designing SDRAM interface. Here is my SDRAM initialization code and circuit;



Now I'm trying to access the U5 that mean the FMC_SDRAM_Bank2 (I'm not using the U4 and it is not on the PCB also resistors too);

```
void SDRAM_Init(void)
{
  FMC_SDRAMInitTypeDef FMCSDRAM;
  FMC_SDRAMTimingInitTypeDef Timing;

  /* Configure the GPIO for FMC SDRAM bank */
  SDRAM_GPIOInit();

  /* Enable FMC clock */
  RCC_AHB3PeriphClockCmd(RCC_AHB3Periph_FMC, ENABLE);

  /* FMC Configuration ---------------------------------------------------------*/
  /* FMC SDRAM Bank Configuration ---------------------------------------------*/
  /* Timing configuration for 90 MHz of SD clock frequency (180 MHz / 2) */
  /* TMRD: 2 Clock cycles */
  Timing.FMC_LoadToActiveDelay          = 2;
  /* TXSR: min = 70 ns (7 x 11.11 ns) */
/* TXSR: min = 67 ns (7 x 11.11 ns) */
  Timing.FMC_ExitSelfRefreshDelay       = 7;
/* TRAS: min = 37 ns  (4 x 11.11 ns) max = 100 k (ns) */
  Timing.FMC_SelfRefreshTime            = 4;
/* TRC: min = 60 ns (6 x 11.11 ns)*/
  Timing.FMC_RowCycleDelay              = 6;
```

```
  /* TWR: min 1 + 7 ns (1 + 1 x 11.11 ns) */
  Timing.FMC_WriteRecoveryTime         = 2;
  /* TRP: 20 ns => 2 x 11.11 ns */
  Timing.FMC_RPDelay                   = 2;
  /* TRCD: 20 ns => 2 x 11.11 ns */
  Timing.FMC_RCDDelay                  = 2;

  /* FMC SDRAM Control configuration ----------------------------------------*/
  FMCSDRAM.FMC_Bank = FMC_Bank2_SDRAM;

  /* Row addressing [9:0] - IS42S16320F */
  FMCSDRAM.FMC_ColumnBitsNumber   = FMC_ColumnBits_Number_10b;
  /* Column addressing [12:0] - IS42S16320F */
  FMCSDRAM.FMC_RowBitsNumber      = FMC_RowBits_Number_13b;

  FMCSDRAM.FMC_SDMemoryDataWidth  = SDRAM_MEMORY_WIDTH;
  FMCSDRAM.FMC_InternalBankNumber = FMC_InternalBank_Number_4;
  FMCSDRAM.FMC_CASLatency         = SDRAM_CAS_LATENCY;
  FMCSDRAM.FMC_WriteProtection    = FMC_Write_Protection_Disable;
  FMCSDRAM.FMC_SDClockPeriod      = SDRAM_CLK_PERIOD;
  FMCSDRAM.FMC_ReadBurst          = FMC_Read_Burst_Disable;
  FMCSDRAM.FMC_ReadPipeDelay      = FMC_ReadPipe_Delay_1;
  // STDR1 = 0x01116363
  FMCSDRAM.FMC_SDRAMTimingStruct  = &Timing;

  /* Initialization of FMC SDRAM Bank-2 */
  FMC_SDRAMInit(&FMCSDRAM);

/* Init sequence for SDRAM-2 */
SDRAM_InitSequence(FMC_Command_Target_bank2, FMC_Bank2_SDRAM);
}

/** ****************************************************************************
  * @brief  Executes the SDRAM memory initialization sequence.
  * @param  None.
  * @retval None.
  * ****************************************************************************
  */
static void SDRAM_InitSequence(uint32_t target_bank, uint32_t fmc_bank)
{
  FMC_SDRAMCommandTypeDef sdCMD;
  uint32_t tempr = 0;

  /* Step 3 ----------------------------------------------------------*/
  /* Configure the clock configuration enable command */
  sdCMD.FMC_CommandMode           = FMC_Command_Mode_CLK_Enabled;
  sdCMD.FMC_CommandTarget         = target_bank;
  sdCMD.FMC_AutoRefreshNumber     = 1;
  sdCMD.FMC_ModeRegisterDefinition = 0;

  /* Wait until the SDRAM controller is ready */
  while(FMC_GetFlagStatus(fmc_bank, FMC_FLAG_Busy) != RESET)
  {}

  /* Send the command */
  FMC_SDRAMCmdConfig(&sdCMD);

  /* Step 4 ----------------------------------------------------------*/
  /* Insert 100ms delay */
  __Delay(100);
```

```
  /* Step 5 --------------------------------------------------------------*/
  /* Configure a PALL (precharge all) command */
  sdCMD.FMC_CommandMode            = FMC_Command_Mode_PALL;
  sdCMD.FMC_CommandTarget          = target_bank;
  sdCMD.FMC_AutoRefreshNumber      = 1;
  sdCMD.FMC_ModeRegisterDefinition = 0;

  /* Wait until the SDRAM controller is ready */
  while(FMC_GetFlagStatus(fmc_bank, FMC_FLAG_Busy) != RESET)
  {}

  /* Send the command */
  FMC_SDRAMCmdConfig(&sdCMD);

  /* Step 6 --------------------------------------------------------------*/
  /* Configure an auto-refresh command */
  sdCMD.FMC_CommandMode            = FMC_Command_Mode_AutoRefresh;
  sdCMD.FMC_CommandTarget          = target_bank;
  sdCMD.FMC_AutoRefreshNumber      = 4;
  sdCMD.FMC_ModeRegisterDefinition = 0;
  /* Wait until the SDRAM controller is ready */
  while(FMC_GetFlagStatus(fmc_bank, FMC_FLAG_Busy) != RESET)
  {}

  /* Send the first command */
  FMC_SDRAMCmdConfig(&sdCMD);
  /* Wait until the SDRAM controller is ready */
  while(FMC_GetFlagStatus(fmc_bank, FMC_FLAG_Busy) != RESET)
  {}

  /* Send the second command */
  FMC_SDRAMCmdConfig(&sdCMD);

  /* Step 7 --------------------------------------------------------------*/
  /* Program the external memory mode register */
  tempr = (uint32_t)SDRAM_MODEREG_BURST_LENGTH_2 |
              SDRAM_MODEREG_BURST_TYPE_SEQUENTIAL |
          SDRAM_MODEREG_CAS_LATENCY_3 |
          SDRAM_MODEREG_OPERATING_MODE_STANDARD |
          SDRAM_MODEREG_WRITEBURST_MODE_SINGLE;

  /* Configure a load Mode register command */
  sdCMD.FMC_CommandMode            = FMC_Command_Mode_LoadMode;
  sdCMD.FMC_CommandTarget          = target_bank;
  sdCMD.FMC_AutoRefreshNumber      = 1;
  sdCMD.FMC_ModeRegisterDefinition = tempr;
  /* Wait until the SDRAM controller is ready */
  while(FMC_GetFlagStatus(fmc_bank, FMC_FLAG_Busy) != RESET)
  {}

  /* Send the command */
  FMC_SDRAMCmdConfig(&sdCMD);

  /* Step 8 --------------------------------------------------------------*/
  /* Set the refresh rate counter */
  /* (15.62 us x Freq) - 20 */ // IS42S16400J = 1386
 /* (7.81 us  x Freq) - 20 */ // IS42S16320F = 682
  /* Set the device refresh counter */
  //FMC_SetRefreshCount(1386);
 FMC_SetRefreshCount(682);
```

```
  /* Wait until the SDRAM controller is ready */
  while(FMC_GetFlagStatus(fmc_bank, FMC_FLAG_Busy) != RESET)
  {}
}

void SDRAM_GPIOInit(void)
{
  GPIO_InitTypeDef gpio;

  /* Enable all gpio port clocks */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB | RCC_AHB1Periph_GPIOC |
                         RCC_AHB1Periph_GPIOD | RCC_AHB1Periph_GPIOE |
                    RCC_AHB1Periph_GPIOF | RCC_AHB1Periph_GPIOG, ENABLE);

  /* Common GPIO configuration */
  gpio.GPIO_Mode  = GPIO_Mode_AF;
  gpio.GPIO_Speed = GPIO_Speed_50MHz;
  gpio.GPIO_OType = GPIO_OType_PP;
// Pull up for 2 SDRAM - TRY-1
  gpio.GPIO_PuPd  = GPIO_PuPd_NOPULL;

  /* GPIOB Configuration */
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource5, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_FMC);

  gpio.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_6;
  GPIO_Init(GPIOB, &gpio);

  /* GPIOC Configuration */
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource0, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource2, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource3, GPIO_AF_FMC);

  gpio.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_2 | GPIO_Pin_3;
  GPIO_Init(GPIOC, &gpio);

  /* GPIOD Configuration */
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource0, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource1, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource8, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource9, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource10, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_FMC);

  gpio.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1  | GPIO_Pin_8  |
                  GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_14 |
                GPIO_Pin_15;
  GPIO_Init(GPIOD, &gpio);

  /* GPIOE Configuration */
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource0, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource1, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource7, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource8, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource9, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource10, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource11, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource12, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource13, GPIO_AF_FMC);
```

```
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource14, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource15, GPIO_AF_FMC);

  gpio.GPIO_Pin = GPIO_Pin_0  | GPIO_Pin_1  | GPIO_Pin_7  |
              GPIO_Pin_8  | GPIO_Pin_9  | GPIO_Pin_10 |
            GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 |
            GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_Init(GPIOE, &gpio);

  /* GPIOF Configuration */
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource0, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource1, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource2, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource3, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource4, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource5, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource11, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource12, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource13, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource14, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOF, GPIO_PinSource15, GPIO_AF_FMC);

  gpio.GPIO_Pin = GPIO_Pin_0  | GPIO_Pin_1  | GPIO_Pin_2  |
              GPIO_Pin_3  | GPIO_Pin_4  | GPIO_Pin_5  |
            GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 |
            GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_Init(GPIOF, &gpio);

  /* GPIOG Configuration */
  GPIO_PinAFConfig(GPIOG, GPIO_PinSource0, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOG, GPIO_PinSource1, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOG, GPIO_PinSource8, GPIO_AF_FMC);
  GPIO_PinAFConfig(GPIOG, GPIO_PinSource15, GPIO_AF_FMC);

  gpio.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_8 | GPIO_Pin_15;
  GPIO_Init(GPIOG, &gpio);
}
```

In the main.c file;

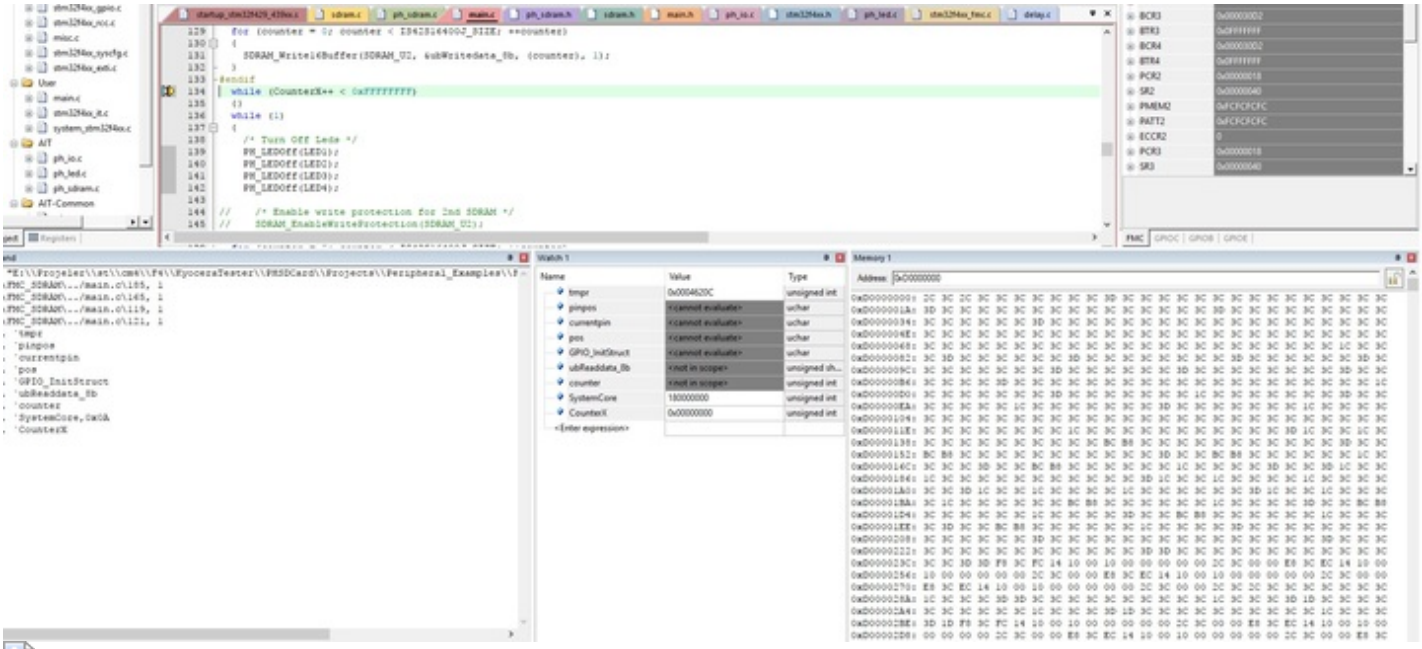
```
#define IS42S16400J_SIZE            0x10000

int main(void)
{
 uint8_t ubWritedata_8b = 0x3C, ubReaddata_8b = 0;
 uint32_t uwReadwritestatus = 0;
 uint32_t counter = 0x0;

 /*!< At this stage the microcontroller clock setting is already configured,
     this is done through SystemInit() function which is called from startup
     file (startup_stm32f429_439xx.s) before to branch to application main.
     To reconfigure the default setting of SystemInit() function, refer to
     system_stm32f4xx.c file
 */

 if (TimingDelay_Init(1000))

 {

   while(1);
```

```
}


SystemCoreClockUpdate();
SystemCore = SystemCoreClock;

 /* SDRAM Initialization */
 SDRAM_Init();


 /* Disable write protection for 2nd SDRAM */
FMC_SDRAMWriteProtectionConfig(FMC_Bank2_SDRAM, DISABLE);


 /* Erase SDRAM memory */
 for (counter = 0x00; counter < (IS42S16400J_SIZE); counter++)
 {
  *(__IO uint8_t*) (SDRAM_BANK2_ADDR + counter) = (uint8_t)0x00;
 }
 /* Write data value to all SDRAM memory */
 for (counter = 0; counter < IS42S16400J_SIZE; counter++)
 {
  *(__IO uint8_t*) (SDRAM_BANK2_ADDR + counter) =\
                          (uint8_t)(ubWritedata_8b);
 }

/* I've added that loop to detect the data situation so I faced the data is changing independently from the code
*/

while (CounterX++ < 0xFFFFFFFF)
{}
 while (1)
 {
  /* Read back SDRAM memory and check content correctness*/
  counter = 0;
  uwReadwritestatus = 0;
  while ((counter < IS42S16400J_SIZE) && (uwReadwritestatus == 0))
  {

    ubReaddata_8b = *(__IO uint8_t *)(SDRAM_BANK2_ADDR + counter);

    if ( ubReaddata_8b != ubWritedata_8b)
    {
     //uwReadwritestatus = 1;

    }
    else
    {

    }
    counter++;
  }

  if(uwReadwritestatus == 0)
  {
  }
  else
  {
  }
```

```
  while (1)
  {

  }
  }
}
```

EDIT 1:

The first image is when the PC stopped at the breakpoint in the picture then I pressed the F10 to go next step that memory situation is indicated in second picture. Data is changing randomly and independently.



EDIT 2:

I've checked the tracks between SDRAM and MCU then found some soldering problems. Now I can read what I wrote but in 45MHz SDRAM clock. There is still missing data in 90MHz SDRAM clock.