main.c 및 user defined function code review

1. SD_read(char *filename)

```
void SD_read(char *filename)
{
  Res = f_mount(&fatfs, Path, 0);
    if(Res != FR_OK){
      xil_printf("\nmount_fail\n");
      return 0;
    }

    Res = f_open(&fil, filename, FA_READ);
    if(Res){
        xil_printf("\nfile_open_fail\n");
        return 0;
    }

    Res = f_lseek(&fil, 0);
    if (Res) {
      xil_printf("\nfseek_fail\n");
      return 0;
    }

    Res = f_read(&fil, buffer, data_size, &NumBytesRead);
    if (Res){
        xil_printf("\ndata_read_fail\n");
        return 0;
    }

    Res = f_close(&fil);

  xil_printf("\nfile_read_success\n");
}
```

자세하게 볼 필요는 없습니다. TFT SoC 강의 자료에 있던 코드를 함수로 만든거에요.

원하는 파일의 포인터를 입력으로 넣으면 SD카드에서 그 파일을 읽어서 buffer에 저장하는 함수입니다.

2. TFTLCD_write_background()

```
void TFTLCD_write_background()
{
  int i, j;

  SD_read(&chess_board);

  for (int i = 0; i < 272; i++){ // loop for vertical
      for (int j = 0; j < 480 / 2; j++){ // loop for horizontal
        // 1
        // Below codes are representing oven number pixel
        Data = (int)buffer[j + 240*i] & 0x0000ffff;
        R = (Data >> 11) & 0x0000001f;
        G = Data & 0x000007E0;
        B = Data & 0x0000001f;
        Data = (B<<11)| G | R;
        Xil_Out32(XPAR_TFTLCD_0_S00_AXI_BASEADDR + (2*j + 480*i)*4, Data);

        // 2
        Data = (int)buffer[j + 240*i] >> 16;
        R = (Data >> 11) & 0x0000001f;
        G = Data & 0x000007E0;
        B = Data & 0x0000001f;
        Data = (B<<11)| G | R;
        Xil_Out32(XPAR_TFTLCD_0_S00_AXI_BASEADDR + (1 + 2*j + 480*i)*4, Data);
        }
    }
}
```

SD_read로 chess_board(기본 체스판) 파일을 읽어오고, 272x480 TFT에 출력시키는 함수입니다.

출력 방식은 자세하게 알 필요는 없고, 출력 영역을 선택하는 방식은 TFTLCD_write_sector에서 설명할게요.

3. TFTLCD_write_sector(char horizontal, int vertical)

```
static int vertical_formation[9] = {0, 34, 68, 102, 136, 170, 204, 238, 272};
static int horizontal_formation[9] = {208, 242, 276, 310, 344, 378, 412, 446, 480};

void TFTLCD_write_sector(char horizontal, int vertical)
{
  int i, j;

  horizontal = (int)'H' - (int)horizontal + 1;

  int horizontal_start  = horizontal_formation[horizontal - 1];
  int horizontal_end    = horizontal_formation[horizontal];
  int vertical_start    = vertical_formation[vertical - 1];
  int vertical_end    = vertical_formation[vertical];

  for (int i = vertical_start + 1; i < vertical_end + 1; i++){ // loop for vertical
      for (int j = horizontal_start / 2 + 1; j < horizontal_end / 2 + 1; j++){ // loop for horizontal
        // 1
      Data = (int)buffer[j + 240 * i] & 0x0000ffff;
      R = (Data >> 11) & 0x0000001f;
      G = Data & 0x000007E0;
      B = Data & 0x0000001f;
      Data = (B<<11)| G | R;
      Xil_Out32(XPAR_TFTLCD_0_S00_AXI_BASEADDR + (2*j + 480*i)*4, Data);
        // 2
      Data = (int)buffer[j + 240 * i] >> 16;
      R = (Data >> 11) & 0x0000001f;
      G = Data & 0x000007E0;
      B = Data & 0x0000001f;
      Data = (B<<11)| G | R;
      Xil_Out32(XPAR_TFTLCD_0_S00_AXI_BASEADDR + (1 + 2*j + 480*i)*4, Data);
      }
    }
}
```

수평 좌표와 수직 좌표를 입력하면 TFT의 해당 영역에 buffer에 들어있는 파일을 출력하는 함수입니다.

처음 짰던 함수는 좌표 입력시 4개의 입력을 받아서 2개의 구간을 입력해주는 방식이었습니다.
하지만 사용자 친화적인 구현을 위해 수평, 수직 좌표만 입력받고 구간은 내부에서 처리하는 방식으로 변경했습니다.

함수에 진입하면 horizontal을 1 ~ 8의 정수로 바꿔주기 위한 처리를 하는데, 이 또한 사용자 친화적인 구현을 위한 코드입니다. (A → 8, B → 7, …. , H → 1 로 변환)

그 후 수평 시작, 끝 점(horizontal_start, horizontal_end), 수직 시작, 끝 점(vertical_start, vertical_end)
을 지정해주고, 해당하는 pixel에 파일을 출력합니다.

2번째 for loop를 보면 horizontal 좌표를 2로 나누어준것을 볼 수 있습니다. 이는 제공받을 코드가 짝수 번째 pixel(**2 * j + 480 * i**)과 홀수 번째 pixel(**1 + 2 * j + 480 * i**)을 번갈아가면서 출력하기 때문에, 절반의 구간에 대해서 for loop를 돌려주면 모든 pixel이 가득차게됩니다.

그리고 이 함수는 background 출력함수와 달리 SD_read()를 넣지 않았습니다. 그 이유는 같은 체스말을 여러 번 출력할 때, 출력할때마다 SD_read()로 memory access를 하면 상당한 delay가 걸리기 때문입니다.

4. move(int start[2], int end[2], char *piece_moved)

```
void move(int start[2], int end[2], char *piece_moved)
{
  SD_read(&chess_board);
  TFTLCD_write_sector(start[0], start[1]);

  SD_read(piece_moved);
  TFTLCD_write_sector(end[0], end[1]);
}
```

TFTLCD_write_sector을 활용해서 체스말을 움직이게하는 함수입니다.

명확히 따지면 시작점을 배경으로 덮고, 끝점에 체스말을 출력하는 방식입니다. 길이가 2인 1차원 배열(start[2], end[2])과 움직이고자 하는 체스말 변수의 포인터를 입력으로 주면 기존에 정의했던 함수를 호출해서 move 동작을 수행합니다.

5. chess_default()

```
void chess_default()
{
  TFTLCD_write_background();

  SD_read(&rook_white);
  TFTLCD_write_sector('H', 1);
  TFTLCD_write_sector('A', 1);

  SD_read(&night_white);
  TFTLCD_write_sector('G', 1);
  TFTLCD_write_sector('B', 1);

  SD_read(&bishop_white);
  TFTLCD_write_sector('F', 1);
  TFTLCD_write_sector('C', 1);

  SD_read(&king_white);
  TFTLCD_write_sector('E', 1);

  SD_read(&queen_white);
  TFTLCD_write_sector('D', 1);

  SD_read(&pawn_white);
  for (char c = 'H'; c >= 'A'; c--)
    TFTLCD_write_sector(c, 2);

  SD_read(&pawn_black);
  for (char c = 'H'; c >= 'A'; c--)
    TFTLCD_write_sector(c, 7);

  SD_read(&rook_black);
  TFTLCD_write_sector('H', 8);
  TFTLCD_write_sector('A', 8);

  SD_read(&night_black);
  TFTLCD_write_sector('G', 8);
  TFTLCD_write_sector('B', 8);

  SD_read(&bishop_black);
  TFTLCD_write_sector('F', 8);
  TFTLCD_write_sector('C', 8);

  SD_read(&king_black);
  TFTLCD_write_sector('E', 8);

  SD_read(&queen_black);
  TFTLCD_write_sector('D', 8);
}
```

체스판을 초기화시키는 함수입니다. 게임 시작하는 시점의 체스말을 setting해줍니다.

6. GetPieceFormation(u8 *piece_formation)

u8은 8bit 자료형 즉 char 입니다.

```
void GetPieceFormation(u8 *piece_formation)
{
  int i = 0;
  char tmp;

  do
  {
    tmp = XUartPs_RecvByte(XPAR_PS7_UART_1_BASEADDR);

    if( (tmp >= '1') && (tmp <= '8') )
    {
      XUartPs_SendByte(XPAR_PS7_UART_1_BASEADDR, tmp);
      piece_formation[i] = tmp;
      i++;
    }
    else if ( (tmp >= 'A') && (tmp <= 'H') || (tmp >= 'a') && (tmp <= 'h'))
    {
      XUartPs_SendByte(XPAR_PS7_UART_1_BASEADDR, tmp);
      piece_formation[i] = tmp;
      i++;
    }
    else
    {
      continue;
    }
```

```
  }
  while( !((tmp == CR) || (i >= NAME_MAX-1)) );

  while(tmp != CR)
    tmp = XUartPs_RecvByte(XPAR_PS7_UART_1_BASEADDR);
}
```

원래 체스말, 시작점, 끝점을 3번에 걸쳐서 입력받았는데, 굳이 그럴 필요가 없었습니다. 그래서 한번의 UART 입력으로 체스말, 시작점, 끝점을 받도록 GetFormation 함수와 GetPiece 함수를 합쳤습니다.

하지만 GetPiece 함수에서 Piece가 어떤 입력인지(pw, kb etc…)에 따라 체스말 변수의 포인터를 가르키는 동작도 했기때문에 이 동작을 위한 SetPiece 함수를 정의했습니다.

다시 돌아와서, 입력받은 문자를 한 글자씩 받아서 piece_formation 포인터가 가르키는 배열에 한 글자씩 넣어주는 함수입니다.

1 ~ 8, A ~ H, a ~ h만 입력받을 수 있고, 숫자도 character로 받기 때문에 추후 정수형으로 type casting 해줘야합니다.

7. SetPiece(u8 *piece)

```
void SetPiece(u8 *piece)
{
  if    (strcmp(piece, "rw") == 0)    piece = &rook_white;
  else if (strcmp(piece, "nw") == 0)    piece = &night_white;
  else if (strcmp(piece, "bw") == 0)    piece = &bishop_white;
  else if (strcmp(piece, "kw") == 0)    piece = &king_white;
  else if (strcmp(piece, "qw") == 0)    piece = &queen_white;
  else if (strcmp(piece, "pw") == 0)    piece = &pawn_white;
  else if (strcmp(piece, "nb") == 0)    piece = &night_black;
  else if (strcmp(piece, "bb") == 0)    piece = &bishop_black;
  else if (strcmp(piece, "kb") == 0)    piece = &king_black;
  else if (strcmp(piece, "qb") == 0)    piece = &queen_black;
  else if (strcmp(piece, "pb") == 0)    piece = &pawn_black;
  else if (strcmp(piece, "pb") == 0)    piece = &pawn_black;

}
```

piece가 가지는 문자열에 따라서 piece를 체스말 변수의 포인터로 바꿔주는 함수입니다.

8. main.c

```
int main()
{
  bool turn_flag = TRUE;
  // TRUE : white turn
  // FALSE : black turn
  int Status;

  //////
  u32 CntrlRegister;
  u8  piece_formation[FORM_MAX] = {0, };
  u8  start_piece[NAME_MAX]   = {0, };
  u8  sel             = 0;
  char  *start_point;
  char  *end_point;

  //////

  CntrlRegister = XUartPs_ReadReg(XPAR_PS7_UART_1_BASEADDR, XUARTPS_CR_OFFSET);

  XUartPs_WriteReg( XPAR_PS7_UART_1_BASEADDR, XUARTPS_CR_OFFSET,
           ((CntrlRegister & ~XUARTPS_CR_EN_DIS_MASK) | XUARTPS_CR_TX_EN | XUARTPS_CR_RX_EN) );

  NumBytesRead = 0;

  InitMsg();
  chess_default();

  while(1)
  {
    PrintChar("Enter the piece and start, end formation(format : pw A2 A3) : ");
    GetPieceFormation(piece_formation);

    char *ptr = strtok(piece_formation, " ");
    start_piece = ptr;
```

```
    ptr = strtok(NULL, " ");
    start_point = ptr;

    ptr = strtok(NULL, " ");
    end_point = ptr;

    SetPiece(start_piece); // Setting pointer of piece wanted to move

    int sp[2] = {start_point[0], int(start_point[1]) - 48};
    int ep[2] = {end_point[0], int(end_point[1]) - 48};


    if (turn_flag && start_piece[1] == 'w')
    {
      move(sp, ep, piece);
      turn_flag = FALSE;
    }
    else if (!turn_flag && start_piece[1] == 'b')
    {
      move(sp, ep, piece);
      turn_flag = TRUE;
    }
    else
    {
      if (turn_flag && start_piece[1] == 'b')
      {
        PrintChar("It is the white turn. Please reenter\n");
        continue;
      }
      else if (!turn_flag && start_piece[1] == 'w')
      {
        PrintChar("It is the black turn.  Please reenter\n");
        continue;
      }
    }


    // Jump to function that process interrupt(push button)
    Status = GicConfigure(INTC_DEVICE_ID);
    if (Status != XST_SUCCESS) {
      xil_printf("GIC Configure Failed\r\n");
      return XST_FAILURE;
    }

    while(TRUE){} // Main loop

    return XST_SUCCESS;
  }

}
```

메인문은 코드가 길기 때문에 동작위주로 설명해보겠습니다.

우선 변수는 piece_formation, start_piece, *start_point, *end_point, turn_flag를 보면 됩니다.

piece_formation은 6번 함수(GetPieceFormation)에서 들어오는 입력을 저장하는 포인터 변수입니다. 그리고 **strtok 함수를 사용해서 piece_formation을 공백기준으로 분리한 후 start_piece, start_point, end_point에 순서대로 넣어줍니다.**

turn_flag는 플레이어가 자기 차례에 맞는 체스말을 골랐는지 체크하기 위해 선언한 변수입니다.

turn_flag : TRUE → white turn
turn_flag : FALSE → black turn

이렇게 동작합니다.

즉, turn_flag가 참일때는 start_piece[1]가 ‘w’ 이어야 하고, 거짓일때는 start_piece[1]가 ‘b’이어야 합니다.

이와 다른 입력이 들어온다면, 조건문을 이용해 다시 입력하라는 메세지를 띄웁니다.

그리고 제일 마지막에 있는 코드는 interrupt process mode로 진입하는 코드입니다.