<Pmod CAN>

[구현 기능]

- Zybo z7 10에서 TMS570 Launchpad로 CAN 을 이용해 데이터를 전송한다.

[준비물]

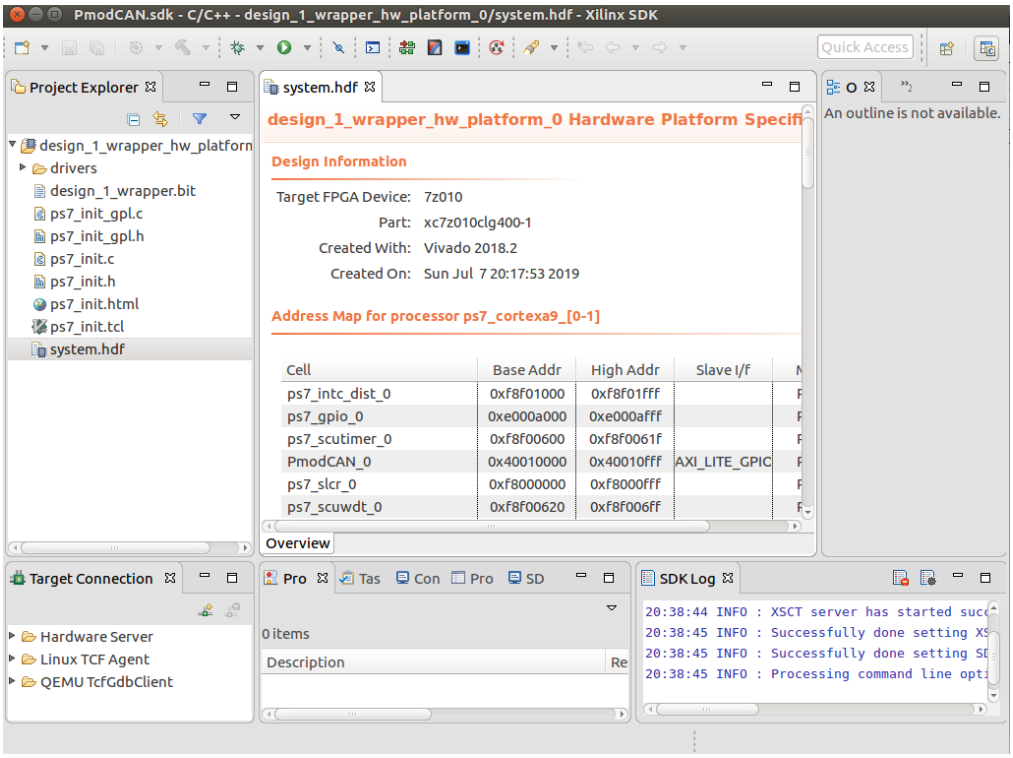
- Zybo z7 10
- Pmod CAN
- tms570lc43 launchpad
- CAN transceiver

[Pmod CAN SDK]

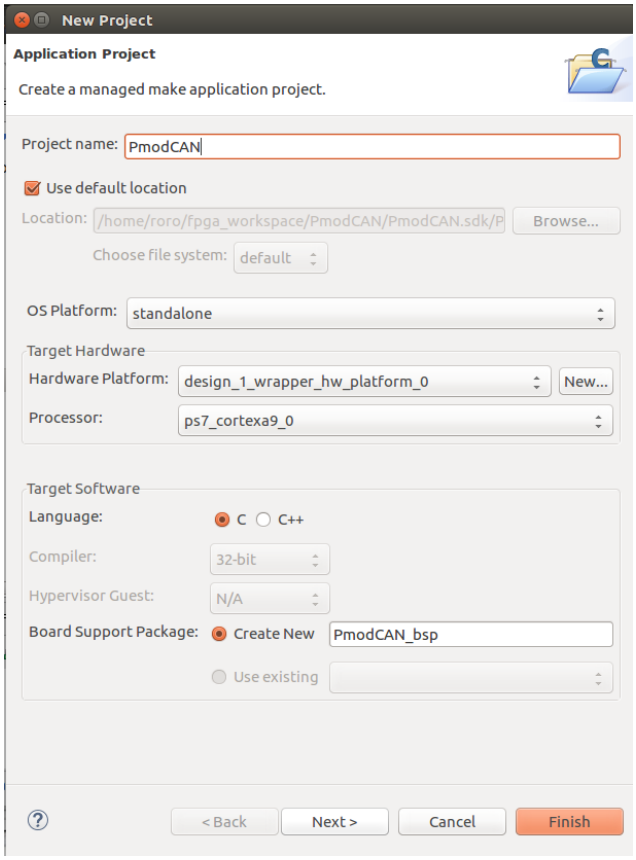
- Digilent tutorial page **10. Tour Xilinx SDK** 부터 참조
https://reference.digilentinc.com/learn/programmable-logic/tutorials/pmod-ips/start

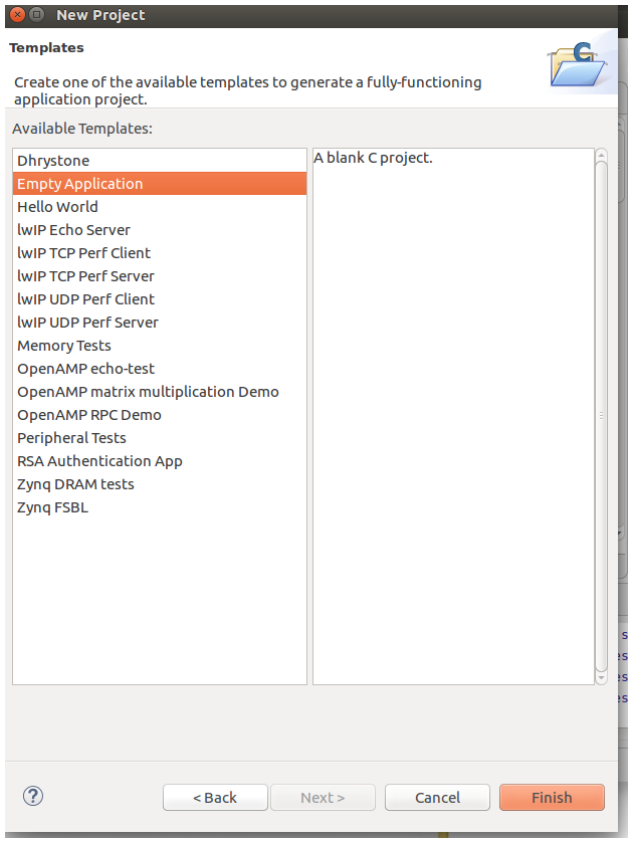
1) SDK 프로젝트 생성

- 지난번에 생성했던 vivado 프로젝트에서 File → Launch SDK 클릭



- Project name, Templates 설정 후 Finish

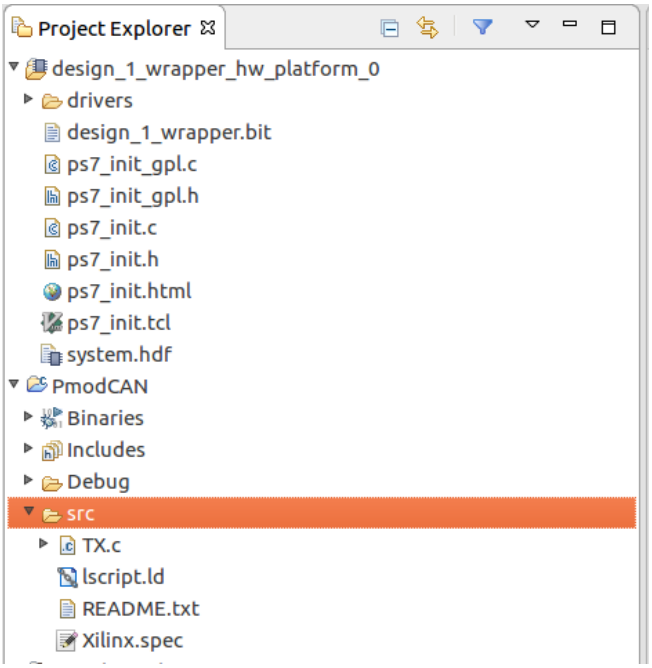




2) Digilent 예제 실행

- Digilent/vivado-library github에서 PmodCAN TX.c예제 참조
https://github.com/Digilent/vivado-library/tree/master/ip/Pmods/PmodCAN_v1_0/drivers/PmodCAN_v1_0/examples

- PmodCAN/src 폴더에 TX.c 파일 추가



3) TX.c 예제 분석

- Zynq Baud Rates : 115200

```
#include "PmodCAN.h"
#include "sleep.h"
#include "xil_cache.h"
#include "xparameters.h"

void DemoInitialize();
void DemoRun();
void DemoCleanup();
void DemoPrintMessage(CAN_Message message);
CAN_Message DemoComposeMessage();
void EnableCaches();
void DisableCaches();

PmodCAN myDevice;

int main(void) {
   DemoInitialize();
   DemoRun();
   DemoCleanup();
   return 0;
}
```

** DemoInitialize() **

```
void DemoInitialize() {
   EnableCaches();
   CAN_begin(&myDevice, XPAR_PMODCAN_0_AXI_LITE_GPIO_BASEADDR,
         XPAR_PMODCAN_0_AXI_LITE_SPI_BASEADDR);
   CAN_Configure(&myDevice, CAN_ModeNormalOperation);
}
```

- 캐시 활성화는(MicroBlaze의 경우에만 해당)

- CAN_begin

1) SPI, GPIO 베이스 주소 할당

2) GPIO in/out 설정

3) CAN SPI Init

```
void CAN_begin(PmodCAN *InstancePtr, u32 GPIO_Address, u32 SPI_Address) {
    InstancePtr->GPIO_addr = GPIO_Address;
    CANConfig.BaseAddress = SPI_Address;

    // 0b1111 for input 0b0000 for output, 0b0001 for pin1 in pin 2 out etc.
    Xil_Out32(InstancePtr->GPIO_addr + 4, 0b1111);
    CAN_SPIInit(&InstancePtr->CANSpi);
}
```

- CAN_Configure

1) CAN Configuration 모드로 수정

2) bit rate configuration

3) CAN buffer/register clear

4) RX 모드 설정

5) normal 모드로 설정

```
void CAN_Configure(PmodCAN *InstancePtr, u8 mode) {
    u8 CNF[3] = {0x86, 0xFB, 0x41};

    // Set CAN control mode to configuration
    CAN_ModifyReg(InstancePtr, CAN_CANCTRL_REG_ADDR, CAN_CAN_CANCTRL_MODE_MASK,
        CAN_ModeConfiguration << CAN_CANCTRL_MODE_BIT);

    // Set config rate and clock for CAN
    CAN_WriteReg(InstancePtr, CAN_CNF3_REG_ADDR, CNF, 3);

    CAN_ClearReg(InstancePtr, 0x00, 12); // Initiate CAN buffer filters and
    CAN_ClearReg(InstancePtr, 0x10, 12); // registers
    CAN_ClearReg(InstancePtr, 0x20, 8);
    CAN_ClearReg(InstancePtr, 0x30, 14);
    CAN_ClearReg(InstancePtr, 0x40, 14);
    CAN_ClearReg(InstancePtr, 0x50, 14);

    // Set the CAN mode for any message type
    CAN_ModifyReg(InstancePtr, CAN_RXB0CTRL_REG_ADDR, 0x64, 0x60);

    // Set CAN control mode to selected mode (exit configuration)
    CAN_ModifyReg(InstancePtr, CAN_CANCTRL_REG_ADDR, CAN_CAN_CANCTRL_MODE_MASK,
          mode << CAN_CANCTRL_MODE_BIT);
}
```

- Bitrate Configuration Digilent 포럼 참조

https://forum.digilentinc.com/topic/18056-zybo-z7-10-pmod-can-bit-rate-problem/

- CAN_SPInit

1) XSpi_CfgInitialize : Xilinx SPI 초기 설정

2) XSpi_SetOptions : SPI 전송 옵션 설정

3) XSpi_SetSlaveSelect : SPI 슬레이브 선택

4) XSpi_Start : SPI 시작

5) XSpi_IntrGlobalDisable : Global 인터럽트 비활성화

```
int CAN_SPIInit(XSpi *SpiInstancePtr) {
    int Status;

    Status = XSpi_CfgInitialize(SpiInstancePtr, &CANConfig,
          CANConfig.BaseAddress);
    if (Status != XST_SUCCESS) {
       return XST_FAILURE;
    }

    // Change these based on your SPI device
    u32 options = (XSP_MASTER_OPTION | XSP_CLK_ACTIVE_LOW_OPTION
          | XSP_CLK_PHASE_1_OPTION) | XSP_MANUAL_SSELECT_OPTION;
    Status = XSpi_SetOptions(SpiInstancePtr, options);
    if (Status != XST_SUCCESS) {
       return XST_FAILURE;
    }

    Status = XSpi_SetSlaveSelect(SpiInstancePtr, 1);
    if (Status != XST_SUCCESS) {
       return XST_FAILURE;
    }

    /*
     * Start the SPI driver so that the device is enabled.
     */
    XSpi_Start(SpiInstancePtr);

    /*
     * Disable Global interrupt to use polled mode operation
     */
    XSpi_IntrGlobalDisable(SpiInstancePtr);

    return XST_SUCCESS;
}
```

** DemoRun **
1) TX0 buffer 가 clear 할 때 까지 대기
2) 메세지 Compose
3) 메세지 프린트 → 디버깅용
4) CAN 인터럽트 레지스터 Flag Clear
5) send 메세지
6) CAN 인터럽트 레지스터 Flag Clear
7) Transmit 이 완료되기를 기다림

```
void DemoRun() {
    CAN_Message TxMessage;
    u8 status;

    xil_printf("Welcome to the PmodCAN IP Core Transmit Demo\r\n");

    while (1) {
        xil_printf("Waiting to send\r\n");
        do {
            status = CAN_ReadStatus(&myDevice);
        } while ((status & CAN_STATUS_TX0REQ_MASK) != 0); // Wait for buffer 0 to
                                                          // be clear

        TxMessage = DemoComposeMessage();

        xil_printf("sending ");
        DemoPrintMessage(TxMessage);

        CAN_ModifyReg(&myDevice, CAN_CANINTF_REG_ADDR, CAN_CANINTF_TX0IF_MASK, 0);

        xil_printf("requesting to transmit message through transmit buffer 0 \
              \r\n");

        CAN_SendMessage(&myDevice, TxMessage, CAN_Tx0);

        CAN_ModifyReg(&myDevice, CAN_CANINTF_REG_ADDR, CAN_CANINTF_TX0IF_MASK, 0);

        do {
            status = CAN_ReadStatus(&myDevice);
            xil_printf("Waiting to complete transmission\r\n");
        } while ((status & CAN_STATUS_TX0IF_MASK) != 0); // Wait for message to
                                                         // transmit successfully

        sleep(1);
    }
}
```

- DemoComposeMessage

1) id = 2

standard mode

data length = 8

```
CAN_Message DemoComposeMessage() {
    CAN_Message message;

    message.id = 0x2;
    message.dlc = 8;
    message.eid = 0x0;
    message.rtr = 0;
    message.ide = 0;
    message.data[0] = 0x01;
    message.data[1] = 0x02;
    message.data[2] = 0x04;
    message.data[3] = 0x08;
    message.data[4] = 0x10;
    message.data[5] = 0x20;
    message.data[6] = 0x40;
    message.data[7] = 0x80;

    return message;
}
```

- CAN_SendMessage

1) Tx 버퍼 선택

2) id, ide, eid, rtr, dlc, data를 버퍼에 저장

3) 1)에서 선택된 버퍼에 load

4) rts 명령어 전송

```
XStatus CAN_SendMessage(PmodCAN *InstancePtr, CAN_Message message,
      CAN_TxBuffer target) {
   u8 data[13];
   u8 i;

   u8 rts_mask;
   u8 load_start_addr;

   switch (target) {
   case CAN_Tx0:
      rts_mask = CAN_RTS_TXB0_MASK;
      load_start_addr = CAN_LOADBUF_TXB0SIDH;
      break;
   case CAN_Tx1:
      rts_mask = CAN_RTS_TXB1_MASK;
      load_start_addr = CAN_LOADBUF_TXB1SIDH;
      break;
   case CAN_Tx2:
      rts_mask = CAN_RTS_TXB2_MASK;
      load_start_addr = CAN_LOADBUF_TXB2SIDH;
      break;
   default:
      return XST_FAILURE;
   }
```

```
   data[0] = (message.id >> 3) & 0xFF; // TXB0 SIDH

   data[1] = (message.id << 5) & 0xE0; // TXB0 SIDL
   data[1] |= (message.ide << 3) & 0x08;
   data[1] |= (message.eid >> 16) & 0x03;

   data[2] = (message.eid >> 8) & 0xFF;
   data[3] = (message.eid) & 0xFF;

   data[4] = (message.rtr << 6) & 0x40;
   data[4] |= (message.dlc) & 0x0F;

   for (i = 0; i < message.dlc; i++)
      data[i + 5] = message.data[i];

   xil_printf("CAN_SendMessage message.dlc: %02x\r\n", message.dlc);
   for (i = 0; i < 5 + message.dlc; i++)
      xil_printf("CAN_SendMessage: %02x\r\n", data[i]);

   CAN_LoadTxBuffer(InstancePtr, load_start_addr, data, message.dlc + 5);
   CAN_RequestToSend(InstancePtr, rts_mask);

   return XST_SUCCESS;
}
```

** DemoCleanup

1) CAN_end 함수

2) Cache 비활성화 (MicroBlaze의 경우에만)

```
void DemoCleanup() {
    CAN_end(&myDevice);
    DisableCaches();
}
```

- CAN_end

```
void CAN_end(PmodCAN *InstancePtr) {
    XSpi_Stop(&InstancePtr->CANSpi);
}
```

- XSpi_Stop

```
int XSpi_Stop(XSpi *InstancePtr)
{
    u32 ControlReg;

    Xil_AssertNonvoid(InstancePtr != NULL);
    Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);

    /*
     * Do not allow the user to stop the device while a transfer is in
     * progress.
     */
    if (InstancePtr->IsBusy) {
        return XST_DEVICE_BUSY;
    }

    /*
     * Disable the device. First disable the interrupts since there is
     * a critical section here because this register is also modified during
     * interrupt context. The device is likely disabled already since there
     * is no transfer in progress, but we do it again just to be sure.
     */
    XSpi_IntrGlobalDisable(InstancePtr);

    ControlReg = XSpi_GetControlReg(InstancePtr);
    XSpi_SetControlReg(InstancePtr, ControlReg & ~XSP_CR_ENABLE_MASK);

    InstancePtr->IsStarted = 0;

    return XST_SUCCESS;
}
```

4) AXI-LITE SPI 레지스터

- LogiCORE IP AXI Serial Peripheral Interface (AXI SPI) (v1.01a) – ds742 참조

Register Overview Table

Table 4 gives a summary of the AXI SPI IP core registers. The Transmit FIFO Occupancy Register and the Receive FIFO Occupancy Register exist only when C_FIFO_EXIST = 1.

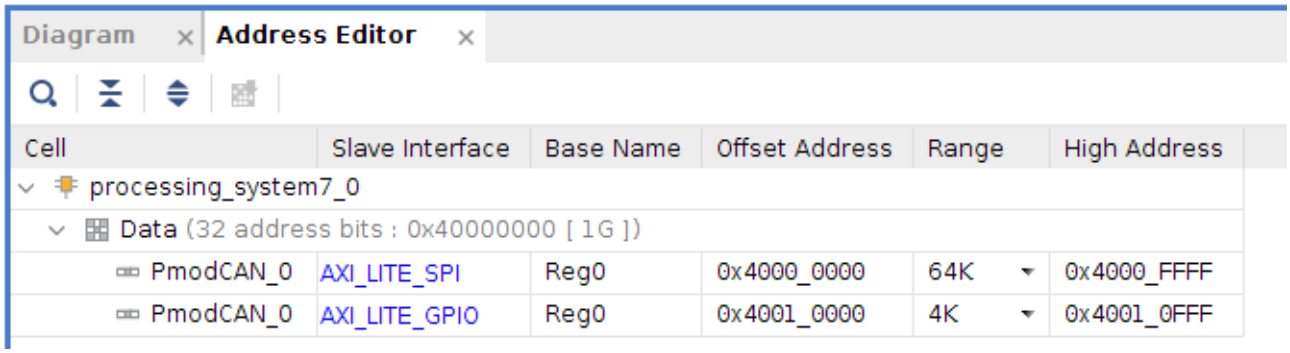
Table 4: **Core Registers**

Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
Core Grouping				
C_BASEADDR + 40	SRR	Write	N/A	Software Reset Register
C_BASEADDR + 60	SPICR	R/W	0x180	SPI Control Register
C_BASEADDR + 64	SPISR	Read	0x25	SPI Status Register
C_BASEADDR + 68	SPIDTR	Write	0x0	SPI Data Transmit Register A single register or a FIFO
C_BASEADDR + 6C	SPIDRR	Read	NA	SPI Data Receive Register A single register or a FIFO
C_BASEADDR + 70	SPISSR	R/W	No slave is selected	SPI Slave Select Register
C_BASEADDR + 74	SPI Transmit FIFO Occupancy Register (1)	Read	0x0	Transmit FIFO Occupancy Register
C_BASEADDR + 78	SPI Receive FIFO Occupancy Register(1)	Read	0x0	Receive FIFO Occupancy Register
Interrupt Controller Grouping				
C_BASEADDR + 1C	DGIER	R/W	0x0	Device Global Interrupt Enable Register
C_BASEADDR + 20	IPISR	R/TOW(2)	0x0	IP Interrupt Status Register
C_BASEADDR + 28	IPIER	R/W	0x0	IP Interrupt Enable Register

Note:

1. Exists only when C_FIFO_EXIST = 1.
2. TOW = Toggle On Write. Writing a 1 to a bit position within the register causes the corresponding bit position in the register to toggle.

- C_BASEADDR은 vivado Address Editor에서 확인



→ 실제로 Hardware design이 AXI Interconnector를 사용하기 때문에 SDK에서 프로그래밍을 할 때에는 AXI-Lite SPI 레지스터를 사용해서 제어해야 함.

- AXI-Lite SPI 레지스터 오프셋을 SDK에서 정의함

```
#define XSP_DGIER_OFFSET 0x1C   /**< Global Intr Enable Reg */
#define XSP_IISR_OFFSET     0x20   /**< Interrupt status Reg */
#define XSP_IIER_OFFSET     0x28   /**< Interrupt Enable Reg */
#define XSP_SRR_OFFSET      0x40   /**< Software Reset register */
#define XSP_CR_OFFSET       0x60   /**< Control register */
#define XSP_SR_OFFSET       0x64   /**< Status Register */
#define XSP_DTR_OFFSET      0x68   /**< Data transmit */
#define XSP_DRR_OFFSET      0x6C   /**< Data receive */
#define XSP_SSR_OFFSET      0x70   /**< 32-bit slave select */
#define XSP_TFO_OFFSET      0x74   /**< Tx FIFO occupancy */
#define XSP_RFO_OFFSET      0x78   /**< Rx FIFO occupancy */
```

- XSpi_Transfer 내부에서 실제로 레지스터에 값을 쓰는 부분

1) DataWidth 검사

2) XSpi_WriteReg 부분에서 BaseAddr를 기준으로 XSP_DTR_OFFSET만큼의 오프셋을 가지고 데이터를 씀

```
StatusReg = XSpi_GetStatusReg(InstancePtr);

while (((StatusReg & XSP_SR_TX_FULL_MASK) == 0) &&
    (InstancePtr->RemainingBytes > 0)) {
    if (DataWidth == XSP_DATAWIDTH_BYTE) {
        /*
         * Data Transfer Width is Byte (8 bit).
         */
        Data = *InstancePtr->SendBufferPtr;
    } else if (DataWidth == XSP_DATAWIDTH_HALF_WORD) {
        /*
         * Data Transfer Width is Half Word (16 bit).
         */
        Data = *(u16 *)InstancePtr->SendBufferPtr;
    } else if (DataWidth == XSP_DATAWIDTH_WORD){
        /*
         * Data Transfer Width is Word (32 bit).
         */
        Data = *(u32 *)InstancePtr->SendBufferPtr;
    }

    XSpi_WriteReg(InstancePtr->BaseAddr, XSP_DTR_OFFSET, Data);
    InstancePtr->SendBufferPtr += (DataWidth >> 3);
    InstancePtr->RemainingBytes -= (DataWidth >> 3);
    StatusReg = XSpi_GetStatusReg(InstancePtr);
```

- XSpi_WriteReg 함수

```
#define XSpi_WriteReg(BaseAddress, RegOffset, RegisterValue) \
    XSpi_Out32((BaseAddress) + (RegOffset), (RegisterValue))

#define XSpi_In32 Xil_In32
#define XSpi_Out32    Xil_Out32
```

```
static INLINE void Xil_Out32(UINTPTR Addr, u32 Value)
{
#ifndef ENABLE_SAFETY
    volatile u32 *LocalAddr = (volatile u32 *)Addr;
    *LocalAddr = Value;
#else
    XStl_RegUpdate(Addr, Value);
#endif
}
```

- 인자로 넘어온 Addr 주소에 Value 값을 씀

[통신 결과]

<TMS570 Launch pad 쪽 출력화면>

: 약 1시간 3000번 이상 전송 완료