Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

CAN 비트 타이밍 requirements

1. bit rate (비트타이밍)
2. bit rate accuracy (비트율 정확도)
3. sample point
4. sample mode
5. Re-Synchronization Jump Width (재동기화 점프폭)

NBT – BUS 에 전송된 초당 비트수

TABLE 4-1: CAN CONTROLLER REGISTER MAP(1)

Lower Address Bits	Higher Order Address Bits							
	0000 xxxx	0001 xxxx	0010 xxxx	0011 xxxx	0100 xxxx	0101 xxxx	0110 xxxx	0111 xxxx
0000	RXF0SIDH	RXF3SIDH	RXM0SIDH	TXB0CTRL	TXB1CTRL	TXB2CTRL	RXB0CTRL	RXB1CTRL
0001	RXF0SIDL	RXF3SIDL	RXM0SIDL	TXB0SIDH	TXB1SIDH	TXB2SIDH	RXB0SIDH	RXB1SIDH
0010	RXF0EID8	RXF3EID8	RXM0EID8	TXB0SIDL	TXB1SIDL	TXB2SIDL	RXB0SIDL	RXB1SIDL
0011	RXF0EID0	RXF3EID0	RXM0EID0	TXB0EID8	TXB1EID8	TXB2EID8	RXB0EID8	RXB1EID8
0100	RXF1SIDH	RXF4SIDH	RXM1SIDH	TXB0EID0	TXB1EID0	TXB2EID0	RXB0EID0	RXB1EID0
0101	RXF1SIDL	RXF4SIDL	RXM1SIDL	TXB0DLC	TXB1DLC	TXB2DLC	RXB0DLC	RXB1DLC
0110	RXF1EID8	RXF4EID8	RXM1EID8	TXB0D0	TXB1D0	TXB2D0	RXB0D0	RXB1D0
0111	RXF1EID0	RXF4EID0	RXM1EID0	TXB0D1	TXB1D1	TXB2D1	RXB0D1	RXB1D1
1000	RXF2SIDH	RXF5SIDH	CNF3	TXB0D2	TXB1D2	TXB2D2	RXB0D2	RXB1D2
1001	RXF2SIDL	RXF5SIDL	CNF2	TXB0D3	TXB1D3	TXB2D3	RXB0D3	RXB1D3
1010	RXF2EID8	RXF5EID8	CNF1	TXB0D4	TXB1D4	TXB2D4	RXB0D4	RXB1D4
1011	RXF2EID0	RXF5EID0	CANINTE	TXB0D5	TXB1D5	TXB2D5	RXB0D5	RXB1D5
1100	BFPCTRL	TEC	CANINTF	TXB0D6	TXB1D6	TXB2D6	RXB0D6	RXB1D6
1101	TXRTSCTRL	REC	EFLG	TXB0D7	TXB1D7	TXB2D7	RXB0D7	RXB1D7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL

→ 레지스터 map 주소, 음영처리된 레지스터는 사용자가 개별로 바꿀 수 있다.

TABLE 4-2: CONTROL REGISTER SUMMARY

Register Name	Address (Hex)	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	POR/RST Value
BFPCTRL	0C	—	—	B1BFS	B0BFS	B1BFE	B0BFE	B1BFM	B0BFM	--00 0000
TXRTSCTRL	0D	—	—	B2RTS	B1RTS	B0RTS	B2RTSM	B1RTSM	B0RTSM	--xx x000
CANSTAT	xE	OPMOD2	OPMOD1	OPMOD0	—	ICOD2	ICOD1	ICOD0	—	100- 000-
CANCTRL	xF	REQOP2	REQOP1	REQOP0	ABAT	OSM	CLKEN	CLKPRE1	CLKPRE0	1110 0111
TEC	1C	Transmit Error Counter (TEC)								0000 0000
REC	1D	Receive Error Counter (REC)								0000 0000
CNF3	28	SOF	WAKFIL	—	—	—	PHSEG2<2:0>			00-- -000
CNF2	29	BTLMODE	SAM	PHSEG1<2:0>			PRSEG2	PRSEG1	PRSEG0	0000 0000
CNF1	2A	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	0000 0000
CANINTE	2B	MERRE	WAKIE	ERRIE	TX2IE	TX1IE	TX0IE	RX1IE	RX0IE	0000 0000
CANINTF	2C	MERRF	WAKIF	ERRIF	TX2IF	TX1IF	TX0IF	RX1IF	RX0IF	0000 0000
EFLG	2D	RX1OVR	RX0OVR	TXBO	TXEP	RXEP	TXWAR	RXWAR	EWARN	0000 0000
TXB0CTRL	30	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
TXB1CTRL	40	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
TXB2CTRL	50	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
RXB0CTRL	60	—	RXM1	RXM0	—	RXRTR	BUKT	BUKT1	FILHIT0	-00- 0000
RXB1CTRL	70	—	RXM1	RXM0	—	RXRTR	FILHIT2	FILHIT1	FILHIT0	-00- 0000

<control register>

PmodCAN register

pmodCAN ref- https://reference.digilentinc.com/reference/pmod/pmodcan/reference-manual

BFPCTRL : PIN CONTROL AND STATUS REGISTER

TXRTSCTRL : TxnRTS PIN CONTROL AND STATUS REGISTER (ADDRESS: 0Dh)

CANSTAT :

CANCTRL :

TEC :

REC :

CNF3 :

CNF2 :

CNF1 :

CANINTE :

CANINTF :

EFLG

TXBxCTRL(x: 0~2) - Message Transmit Registers
:TRANSMIT BUFFER x CONTROL REGISTER (ADDRESS: 30h, 40h, 50h, 시작주소)
메시지 중단 플래그, 메시지 손실 중재, 전송 오류 감지, 메시지 전송 요청, 전송 버퍼 우선 순위

RXB0CTRL :

RXB1CTRL :

PmodCAN.h

함수 프로토타입

```
1. void CAN_begin(PmodCAN *InstancePtr, u32 GPIO_Address, u32 SPI_Address);
: CAN SPI 초기화(to CAN_SPIInit()), gpio 메모리위치에 대한 출력작업

2. void CAN_end(PmodCAN *InstancePtr);
:

3. int CAN_SPIInit(XSpi *SPIInstancePtr);
: CAN SPI 초기화 함수

4. u8 CAN_ReadByte(PmodCAN *InstancePtr);

5. void CAN_WriteByte(PmodCAN *InstancePtr, u8 cmd);

6. void CAN_WriteSPI(PmodCAN *InstancePtr, u8 reg, u8 *wData, int nData);

7. void CAN_ReadSPI(PmodCAN *InstancePtr, u8 reg, u8 *rData, int nData);
8. void CAN_SetRegisterBits(PmodCAN *InstancePtr, u8 reg, u8 mask, bool fValue);
9. u8 CAN_GetRegisterBits(PmodCAN *InstancePtr, u8 bRegisterAddress, u8 bMask);
10. void CAN_ModifyReg(PmodCAN *InstancePtr, u8 reg, u8 mask, u8 value);
11. void CAN_WriteReg(PmodCAN *InstancePtr, u8 reg, u8 *data, u32 nData);
12. void CAN_ClearReg(PmodCAN *InstancePtr, u8 reg, u32 nData);
13. void CAN_LoadTxBuffer(PmodCAN *InstancePtr, u8 start_addr, u8 *data, u32 nData);
14. void CAN_RequestToSend(PmodCAN *InstancePtr, u8 mask);
15. void CAN_ReadRxBuffer(PmodCAN *InstancePtr, u8 start_addr, u8 *data, u32 nData);
16. void CAN_ReadReg(PmodCAN *InstancePtr, u8 reg, u8 *data, u32 nData);
17. u8 CAN_ReadStatus(PmodCAN *InstancePtr);
18. u8 CAN_RxStatus(PmodCAN *InstancePtr);
19. void CAN_Configure(PmodCAN *InstancePtr, u8 mode); // This function is missing
                    // some potential parameters
20. XStatus CAN_SendMessage(PmodCAN *InstancePtr, CAN_Message message,
  CAN_TxBuffer target);
21. XStatus CAN_ReceiveMessage(PmodCAN *InstancePtr, CAN_Message *MessagePtr,
  CAN_RxBuffer target);
```

xspi.h

함수 프로토타입

```
int XSpi_Initialize(XSpi *InstancePtr, u16 DeviceId);
XSpi_Config *XSpi_LookupConfig(u16 DeviceId);


 // Functions, in xspi.c
int XSpi_CfgInitialize(XSpi *InstancePtr, XSpi_Config * Config, u32 EffectiveAddr);

int XSpi_Start(XSpi *InstancePtr);
int XSpi_Stop(XSpi *InstancePtr);

void XSpi_Reset(XSpi *InstancePtr);

int XSpi_SetSlaveSelect(XSpi *InstancePtr, u32 SlaveMask);
u32 XSpi_GetSlaveSelect(XSpi *InstancePtr);

int XSpi_Transfer(XSpi *InstancePtr, u8 *SendBufPtr, u8 *RecvBufPtr, unsigned int ByteCount);

void XSpi_SetStatusHandler(XSpi *InstancePtr, void *CallBackRef, XSpi_StatusHandler FuncPtr);
void XSpi_InterruptHandler(void *InstancePtr);

 // Functions for selftest, in xspi_selftest.c
int XSpi_SelfTest(XSpi *InstancePtr);

// Functions for statistics, in xspi_stats.c
void XSpi_GetStats(XSpi *InstancePtr, XSpi_Stats *StatsPtr);
void XSpi_ClearStats(XSpi *InstancePtr);

 //Functions for options, in xspi_options.c
int XSpi_SetOptions(XSpi *InstancePtr, u32 Options);
u32 XSpi_GetOptions(XSpi *InstancePtr);
```

vivado 설계 → sdk

1. vivado

https://github.com/KOITT2/RC_Car/blob/master/experiment/doc/Pmod_CAN_Control_with_Zybo.pdf

2.

RX main

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

분석

1. DemoInitialize() 함수 분석

zynq ref - https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf
AXI ref- https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf
AXI4-Lite ref- https://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif/v3_0/pg155-axi-lite-ipif.pdf

```
void DemoInitialize(a,b,c) {      // 캔 구성 초기화 및 설정
 EnableCaches();
 CAN_begin(&myDevice, XPAR_PMODCAN_0_AXI_LITE_GPIO_BASEADDR,XPAR_PMODCAN_0_AXI_LITE_SPI_BASEADDR);

 CAN_Configure(&myDevice, CAN_ModeNormalOperation);
 //  설정 ( spi transfer을 사용하여 pmod can을 구성 및 통신할 pmodCAN 객체설정 )
}
```

1.1 CAN_begin(1,2,3) 함수 (DemoInitialize함수→ sub)

```
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); // 4001 0004, 0b1111
 CAN_SPIInit(&InstancePtr->CANSpi);
}
```

인자1. &myDevice

PmodCAN myDevice; // pmodCAN 객체

: pmodCAN은 Gpio peripheral 제어를 spi 인터페이스로 제어한다.

```
typedef struct PmodCAN {
   u32 GPIO_addr;
   XSpi CANSpi;
} PmodCAN;
```

인자2. XPAR_PMODCAN_0_AXI_LITE_GPIO_BASEADDR

xparameters.h : #define XPAR_PMODCAN_0_AXI_LITE_GPIO_BASEADDR 0x40010000 라고 GPIO레지스터 주소가 설정되어있다. 이는 우리가 디자인한 ZYNQ의 address Map이고, 비바도에서 export 해준 하드웨어 정보가 들어 있다. 아래 그림은 system.hdf 파일에 있다. (hw_platform), **인자3. XPAR_PMODCAN_0_AXI_LITE_SPI_BASEADDR 는 0x40000000 임.**

PmodCAN_0	0x40000000	0x4000ffff	AXI_LITE_SPI	REGISTER
PmodCAN_0	0x40010000	0x40010fff	AXI_LITE_GPIO	REGISTER

아래 두 그림은 Vivado 블럭디자인 과 zynq reference 메뉴얼의 AddressMap 이다.
Pmod CAN은 axi bus를 통해 zynq processer와 연결되어 있다. 이는 AXI_GP0 의 물리 주소로써 리눅스 어플리케이션에서 직접 access할 수 있는 가상주소에 mapping 하는데 사용될 거라 생각한다.

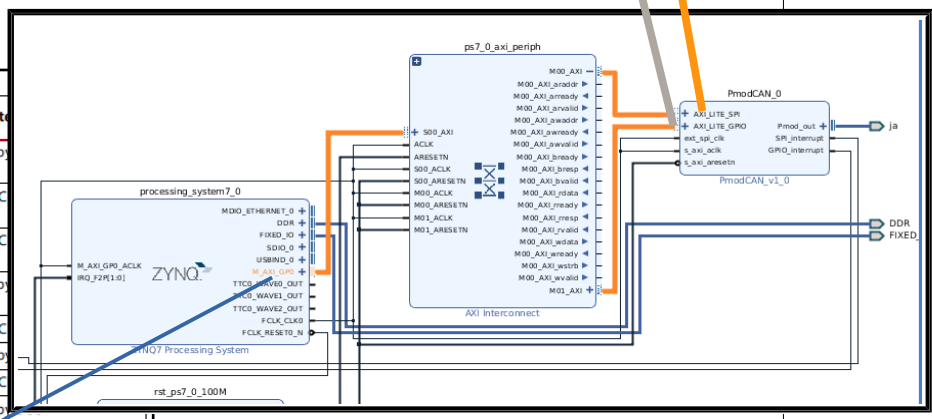


Table 4-1: **System-Level Address Map**

Address Range	CPUs and ACP	AXI_HP	Other Bus Masters[1]	Note
0000_0000 to 0003_FFFF[2]	OCM	OCM	OCM	Address not filtered by mapped low
	DDR	OCM	OCM	Address filtered by SC mapped low
	DDR			Address filtered by SC mapped low
				Address not filtered by not mapped low
0004_0000 to 0007_FFFF	DDR			Address filtered by SC
				Address not filtered by
0008_0000 to 000F_FFFF	DDR	DDR	DDR	Address filtered by SC
		DDR	DDR	Address not filtered by
0010_0000 to 3FFF_FFFF	DDR	DDR	DDR	Accessible to all interconnect masters
4000_0000 to 7FFF_FFFF	PL		PL	General Purpose Port #0 to the PL, M_AXI_GP0
8000_0000 to BFFF_FFFF	PL		PL	General Purpose Port #1 to the PL, M_AXI_GP1
E000_0000 to E02F_FFFF	IOP		IOP	I/O Peripheral registers, see Table 4-6
E100_0000 to E5FF_FFFF	SMC		SMC	SMC Memories, see Table 4-5
F800_0000 to F800_0BFF	SLCR		SLCR	SLCR registers, see Table 4-3
F800_1000 to F880_FFFF	PS		PS	PS System registers, see Table 4-7
F890_0000 to F8F0_2FFF	CPU			CPU Private registers, see Table 4-4
FC00_0000 to FDFF_FFFF[4]	Quad-SPI		Quad-SPI	Quad-SPI linear address for linear mode
FFFC_0000 to FFFF_FFFF[2]	OCM	OCM	OCM	OCM is mapped high
				OCM is not mapped high

1.1.1 Xil_Out32()함수, (DemoInitialize함수 → CAN_begin → sub)

→ 메모리 위치에 대한 출력 작업을 함.

------ ------ ------ ------ ------ ------ ------ code ------ ------ ------ ------ ------ ------ ------ ------ ------

```
static INLINE void Xil_Out32(UINTPTR Addr, u32 Value)
   // InstancePtr->GPIO_addr + 4= 4001 0004, 0b1111
{
#ifndef ENABLE_SAFETY      // ENABLE_SAFETY 가 define으로 정의 되어있지 않으면 발생함.
        volatile u32 *LocalAddr = (volatile u32 *)Addr; // 4001 0004
        *LocalAddr = Value;          // gpio_address(0x40010000) + 4  주소에 0b1111을 넣음.
#else                       // ENABLE_SAFETY 가 define으로 정의되어 있으면 발생
        XStl_RegUpdate(Addr, Value);
#endif
}
```

→ gpio address 0x4001000 + 4 → memory 최소 단위 4byte를 감안해서 인듯하고, 0b1111을 넣어 줌으로써 gpio를 입력으로 설정하였다. 0b1111은 입력 0b0000은 출력, 0b0001은 pin1은 입력이고 pin2는 출력으로 설정하는것.

* volatile 변수 - 사용할 때 항상 메모리에 접근한다. 즉, 이 변수는 언제든지 값이 바뀔 수 있으니까 항상 메모리에 접근하라고 컴파일러에게 알려주는 것이다.
* static 함수 - static으로 선언해놓으면 그 파일 내에서만 사용하는 것. 다른 c파일에서 함수명이 같으면 링커에러나 이상한 동작이 일어날 수 있고, 또한 외부에 인터페이스를 노출하지 않는다는 점에서 사용한다.
* static 변수 - 지역변수처럼 선언된 함수내에서만 사용이 가능하며, 단 한번만 초기화를 할 뿐 전역 변수처럼 프로그램이 종료될 때까지 메모리공간에 존재하게된다.

1.1.2 CAN_SPIInit() 함수, (DemoInitialize함수 → CAN_begin → sub)

```
-------------------------      Xspi  signal    ----------------------------------
typedef struct {
        XSpi_Stats Stats;      /**< Statistics */
        u32 BaseAddr;          /**< Base address of device (IPIF) */
        int IsReady;           /**< Device is initialized and ready */
        int IsStarted;         /**< Device has been started */
        int HasFifos;          /**< Device is configured with FIFOs or not */
        u32 SlaveOnly;         /**< Device is configured to be slave only */
        u8 NumSlaveBits;       /**< Number of slave selects for this device */
        u8 DataWidth;          /**< Data Transfer Width 8 or 16 or 32 */
        u8 SpiMode;            /**< Standard/Dual/Quad mode */
        u32 SlaveSelectMask;   /**< Mask that matches the number of SS bits */
        u32 SlaveSelectReg;    /**< Slave select register */

        u8 *SendBufferPtr;     /**< Buffer to send  */
        u8 *RecvBufferPtr;     /**< Buffer to receive */
        unsigned int RequestedBytes; /**< Total bytes to transfer (state) */
        unsigned int RemainingBytes; /**< Bytes left to transfer (state) */
        int IsBusy;            /**< A transfer is in progress (state) */

        XSpi_StatusHandler StatusHandler; /**< Status Handler */
        void *StatusRef;       /**< Callback reference for status handler */
        u32 FlashBaseAddr;     /**< Used in XIP Mode */
        u8 XipMode;             /**< 0 if Non-XIP, 1 if XIP Mode */
} Xspi;  //spi 인터페이스 신호
```

```
------------------------------      code      ----------------------------------
int CAN_SPIInit(XSpi *SpiInstancePtr) {
 int Status;  // 장치드라이버의 공통의 상태 코드를 전송한다.

 Status = XSpi_CfgInitialize(SpiInstancePtr, &CANConfig,
   CANConfig.BaseAddress);   // spi초기화 함수
 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);
// 장치 구성을 변경할 때, 다른 슬레이브로부터 공유된 곳을 보호함. (세마포어)
```

```
 if (Status != XST_SUCCESS) {
  return XST_FAILURE;
 }

 Status = XSpi_SetSlaveSelect(SpiInstancePtr, 1);  //SS 슬레이브 select 함수( 슬레이브들 중 장치 선택 )
 if (Status != XST_SUCCESS) {
  return XST_FAILURE;
 }

 // Start the SPI driver so that the device is enabled. 장치를 시작한다.
 XSpi_Start(SpiInstancePtr)

 // Disable Global interrupt to use polled mode operation 폴링모드에서 인터럽트 사용 안함.
 XSpi_IntrGlobalDisable(SpiInstancePtr);

 return XST_SUCCESS;`
}
```

---------- **code** ----------

1.1.2.1 Xspi_CfgInitialize() 함수

(DemoInitialize함수 → CAN_begin → CAN_SPIInit()→ sub)

```
int XSpi_CfgInitialize(XSpi *InstancePtr, XSpi_Config *Config, u32 EffectiveAddr)
  //SpiInstancePtr(pmod_spi), &CANConfig(config_spi), CANConfig.BaseAddress(ps_spi_base address)
{     // spi_config struct로 구성을 변경할때 몇몇레지스터를 바꿔줘야함으로 또하나의 구조체를 생성하여 넣어주는 것.
       u8  Buffer[3];
       u32 ControlReg;

       Xil_AssertNonvoid(InstancePtr != NULL);

       /*
        * If the device is started, disallow the initialize and return a status
        * indicating it is started.  This allows the user to stop the device
        * and reinitialize, but prevents a user from inadvertently
        * initializing.
        */
       if (InstancePtr->IsStarted == XIL_COMPONENT_IS_STARTED) {  // spi start 레지스터
              return XST_DEVICE_IS_STARTED;
       }

       /*
        * Set some default values.
        */
       InstancePtr->IsStarted = 0;
       InstancePtr->IsBusy = FALSE;

       InstancePtr->StatusHandler = StubStatusHandler;

       InstancePtr->SendBufferPtr = NULL;
       InstancePtr->RecvBufferPtr = NULL;
       InstancePtr->RequestedBytes = 0;
       InstancePtr->RemainingBytes = 0;
       InstancePtr->BaseAddr = EffectiveAddr;
       InstancePtr->HasFifos = Config->HasFifos;
       InstancePtr->SlaveOnly = Config->SlaveOnly;
       InstancePtr->NumSlaveBits = Config->NumSlaveBits;
       if (Config->DataWidth == 0) {
              InstancePtr->DataWidth = XSP_DATAWIDTH_BYTE;
       } else {
              InstancePtr->DataWidth = Config->DataWidth;
       }

       InstancePtr->SpiMode = Config->SpiMode;

       InstancePtr->FlashBaseAddr = Config->AxiFullBaseAddress;
       InstancePtr->XipMode = Config->XipMode;

       InstancePtr->IsReady = XIL_COMPONENT_IS_READY;

       /*
        * Create a slave select mask based on the number of bits that can
        * be used to deselect all slaves, initialize the value to put into
```

```
         * the slave select register to this value.
         */
        InstancePtr->SlaveSelectMask = (1 << InstancePtr->NumSlaveBits) - 1;
        InstancePtr->SlaveSelectReg = InstancePtr->SlaveSelectMask;

        /*
         * Clear the statistics for this driver.
         */
        InstancePtr->Stats.ModeFaults = 0;
        InstancePtr->Stats.XmitUnderruns = 0;
        InstancePtr->Stats.RecvOverruns = 0;
        InstancePtr->Stats.SlaveModeFaults = 0;
        InstancePtr->Stats.BytesTransferred = 0;
        InstancePtr->Stats.NumInterrupts = 0;

    if(Config->Use_Startup == 1) {
                /*
                 * Perform a dummy read this is used when startup block is
                 * enabled in the hardware to fix CR #721229.
                 */
                //control regiset선택 spi_base_address + control register offset
                ControlReg = XSpi_GetControlReg(InstancePtr);

                // 컨트롤레지스터에 리셋 넣어줌.
                ControlReg |= XSP_CR_TXFIFO_RESET_MASK | XSP_CR_RXFIFO_RESET_MASK |
                              XSP_CR_ENABLE_MASK | XSP_CR_MASTER_MODE_MASK ;
                XSpi_SetControlReg(InstancePtr, ControlReg); // 초기화

                /*
                 * Initiate Read command to get the ID. This Read command is for
                 * Numonyx flash.
                 *
                 * NOTE: If user interfaces different flash to the SPI controller
                 * this command need to be changed according to target flash Read
                 * command.
                 */
                Buffer[0] = 0x9F;
                Buffer[1] = 0x00;
                Buffer[2] = 0x00;

                /* Write dummy ReadId to the DTR register */
                XSpi_WriteReg(InstancePtr->BaseAddr, XSP_DTR_OFFSET, Buffer[0]);
                XSpi_WriteReg(InstancePtr->BaseAddr, XSP_DTR_OFFSET, Buffer[1]);
                XSpi_WriteReg(InstancePtr->BaseAddr, XSP_DTR_OFFSET, Buffer[2]);

                /* Master Inhibit enable in the CR */
                ControlReg = XSpi_GetControlReg(InstancePtr);
                ControlReg &= ~XSP_CR_TRANS_INHIBIT_MASK;
                XSpi_SetControlReg(InstancePtr, ControlReg);

                /* Master Inhibit disable in the CR */
                ControlReg = XSpi_GetControlReg(InstancePtr);
                ControlReg |= XSP_CR_TRANS_INHIBIT_MASK;
                XSpi_SetControlReg(InstancePtr, ControlReg);

                /* Read the Rx Data Register */
                XSpi_ReadReg(InstancePtr->BaseAddr, XSP_DRR_OFFSET);
                XSpi_ReadReg(InstancePtr->BaseAddr, XSP_DRR_OFFSET);
        }

        /*
         * Reset the SPI device to get it into its initial state. It is expected
         * that device configuration will take place after this initialization
         * is done, but before the device is started.
         */
        XSpi_Reset(InstancePtr);

        return XST_SUCCESS;
}
```

→ spi 인터페이스에 대해 공부하고 분석해야 함.

샘 spi정리 https://github.com/KOITT2/RC_Car/blob/master/experiment/doc/SPI_Comm.pdf

1.2 CAN_Configure(a,b)
캔모듈(MCP25625) http://ww1.microchip.com/downloads/en/DeviceDoc/20005282B.pdf

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

 /* Set CAN control mode to configuration
    CAN 제어 모드를 구성으로 설정
 1. 수정 레지스터 전송 SPI 명령 (0x05)
 2. MCP_CANCTRL (0x0F) 컨트롤의 주소를 보냅니다.     // CANCTRL 처음주소
 3. 준비된 것을 얻기 위해 가면 보내기 (0x80)
 4. 모듈을 구성 모드 (0x80)에 놓으라는 명령을 보냅니다.*/
 CAN_ModifyReg(InstancePtr, CAN_CANCTRL_REG_ADDR, CAN_CAN_CANCTRL_MODE_MASK, CAN_ModeConfiguration);

 /* Set config rate and clock for CAN
    CAN의 설정 속도 및 클록 설정
 1. 쓰기 SPI 명령 보내기 (0x02)
 2. 수정하고자하는 레지스터의 주소와 설정 값을 보낸다.
 3. 세 가지 레지스터는 다양한 설정으로 이러한 방식으로 조작됩니다. 자세한 내용은 MCP25625 데이터 시트의 섹션 4.4 (47 페이지)를
참조하십시오 . 다음 세 명령은 CAN 속도가 20 MHz 인 250 kBPS의 CAN 속도를 설정합니다 .
    0x41로 설정된 CNF1 (0x2A)
    0xFB로 설정된 CNF2 (0x29)
    0x86으로 설정된 CNF3 (0x28)*/
 CAN_WriteReg(InstancePtr, CAN_CNF3_REG_ADDR, CNF, 3);


/* Initiate는 필터와 레지스터를 버퍼링 할 수 있습니다.
1. 수신 필터를 표준 또는 확장 식별자로 설정
 1.1. 모든 표준 입력 및 데모 코드에서 레지스터 0x00에서 0x0B, 레지스터 0x10에서 0x1B, 레지스터 0x20에서 0x27을 0x00의 값으
로 설정합니다. 전송 레지스터 플래그와 설정은 레지스터 0x30 ~ 0x3D, 0x40 ~ 0x4D 및 0x50 ~ 0x5D 레지스터를 0x00 값으로 설
정하여 모두 지 웁니다. 다음 단계를 수행하여 각 레지스터에 대해이 작업을 수행 할 수 있습니다.
 1.2. 쓰기 SPI 명령 보내기 (0x02)
 1.3. 관심 레지스터 주소 보내기
 1.4. 쓰여질 값 보내기 (0x00) */
 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
     모든 메시지 유형에 대해 CAN 모드 설정
 1.수정 레지스터 전송 SPI 명령 (0x05)
 2.제어 RXB0CNTRL (0x60)의 주소를 전송하십시오.   R XB0CTRL 시작주소
 3.준비된 것을 얻기 위해 가면 보내기 (0x64)
 4.모든 메시지 유형 (0x60)을 받아들이도록 실제 명령 보내기 */
 CAN_ModifyReg(InstancePtr, CAN_RXB0CTRL_REG_ADDR, 0x64, 0x60);

 /* Set CAN control mode to selected mode (exit configuration)
   CAN 제어 모드를 일반 모드로 설정하십시오.
 1.수정 레지스터 전송 SPI 명령 (0x05)
 2.MCP_CANCTRL (0x0F) 컨트롤의 주소를 보냅니다.
 3.준비된 것을 얻기 위해 가면 보내기 (0x80)
 4.구성 모드 (0x00)에 넣으려면 실제 명령을 보냅니다. */
 CAN_ModifyReg(InstancePtr, CAN_CANCTRL_REG_ADDR, CAN_CAN_CANCTRL_MODE_MASK, mode << CAN_CANCTRL_MODE_BIT);
}
```

a. &myDevice = Pmodcan 객체 (

```
typedef struct PmodCAN {
   u32 GPIO_addr;
   XSpi CANSpi;
} PmodCAN;
```

)

b. CAN_ModeNormalOperation

: normal mode

일반 모드는 MCP25625의 표준 작동 모드이다. 이 모드에서 장치는 모든 버스 메시지를 능동적으로 모니터링하고 Acknowledge 비트, 오류 프레임 등을 생성한다. 이는 MCP25625가 CAN 버스를 통해 메시지를 전송하는 유일한 모드이기도하다. CAN 컨트롤러와 CAN 트랜시버는 모두 Normal 모드 여야한다.

----------bps 설정 ----------

CNF1, CNF2, CNF3 레지스터로 설정한다.

REGISTER 4-26: CNF1: CONFIGURATION 1 REGISTER (ADDRESS: 2Ah)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
bit 7							bit 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **SJW<1:0>:** Synchronization Jump Width Length bits
11 = Length = 4 x T_Q
10 = Length = 3 x T_Q
01 = Length = 2 x T_Q
00 = Length = 1 x T_Q

bit 5-0 **BRP<5:0>:** Baud Rate Prescaler bits
$T_Q = 2 \times (BRP<5:0> + 1)/F_{OSC}$

CNF1 = 0x41
0100 0001
FOSC = 20Mhz 오실레이터 주파수

→ Length = 2*TQ
→ TQ = 2*(1+1)/20Mhz = 0.2us

REGISTER 4-27: CNF2: CONFIGURATION 2 REGISTER (ADDRESS: 29h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BTLMODE	SAM	PHSEG1<2:0>			PRSEG2	PRSEG1	PRSEG0
bit 7							bit 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **BTLMODE:** PS2 Bit Time Length bit
1 = Length of PS2 is determined by the PHSEG2<2:0> bits of CNF3
0 = Length of PS2 is the greater of PS1 and IPT (2 T_Q)

bit 6 **SAM**: Sample Point Configuration bit
1 = Bus line is sampled three times at the sample point
0 = Bus line is sampled once at the sample point

bit 5-3 **PHSEG1<2:0>:** PS1 Length bits
(PHSEG1<2:0> + 1) x T_Q

bit 2-0 **PRSEG<2:0>:** Propagation Segment Length bits
(PRSEG<2:0> + 1) x T_Q

CNF2 = 0xFB
1111 1011

→ PHSEG1 = 8TQ
→ PRSEG = 4TQ

REGISTER 4-28: CNF3: CONFIGURATION 3 REGISTER (ADDRESS: 28h)

R/W-0	R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
SOF	WAKFIL	—	—	—	PHSEG2<2:0>		
bit 7							bit 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **SOF:** Start-of-Frame Signal bit
If CLKEN (CANCTRL<2>) = 1:
1 = CLKOUT pin is enabled for SOF signal
0 = CLKOUT pin is enabled for clock out function
If CLKEN (CANCTRL<2>) = 0:
Bit is don't care.

bit 6 **WAKFIL:** Wake-up Filter bit
1 = Wake-up filter is enabled
0 = Wake-up filter is disabled

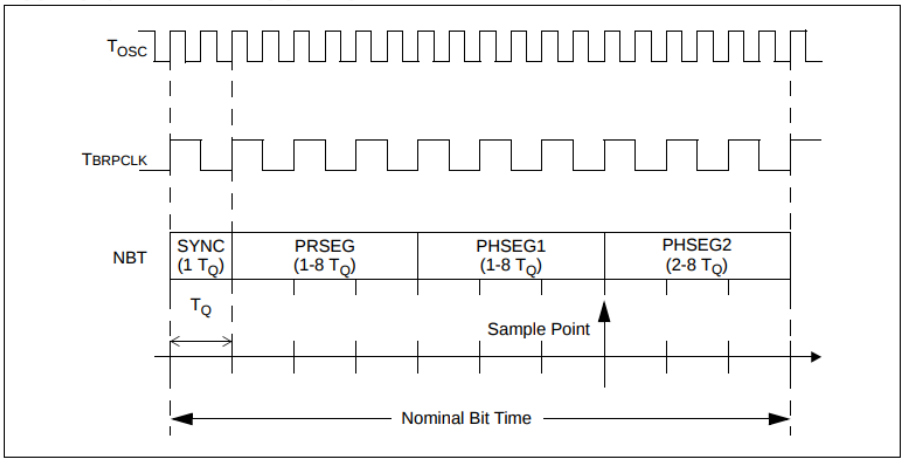
bit 5-3 **Unimplemented:** Read as '0'

bit 2-0 **PHSEG2<2:0>:** PS2 Length bits
(PHSEG2<2:0> + 1) x T_Q
Minimum valid setting for PS2 is 2 T_Q.

CNF3 = 0x86
1000 0110

→ CLKOUT pin enable (SOF)
→ PHSEG2 = 7TQ

FIGURE 3-10: ELEMENTS OF A NOMINAL BIT TIME



□ Sync_Seg : Bus에서 다양한 nodes를 동기화하는데 사용
□ Prop_Seg : 어떤 물리적 지연 보상 (physical bus 와 internal CAN node 전달 지연)
□ Phase_Seg1, Phase_Seg2 : phase edge 오류를 위한 보정에 사용. 이러한 구분은 재동기화 시 늘어난 시간을 단축

: PRSEG + PHSEG1 + PHSEG2 = 4 + 8 + 7 = 19 TQ
+ SYNC = 20TQ

TQ= 0.2us → 20TQ = 4us → 250 kbps

1.2.1 CAN_ModifyReg(

----------code----------

```
void CAN_ModifyReg(PmodCAN *InstancePtr, u8 reg, u8 mask, u8 value) {

 u8 buf[4] = {CAN_MODIFY_REG_CMD, reg, mask, value};
 XSpi_Transfer(&InstancePtr->CANSpi, buf, NULL, 4);
}
```

DemoInitialize 함수

: &mydevice 로 pmodcan의 객체를 만들어 gpio address에 맵핑하고, 그안에 0b1111을 집어 넣었다. 그리고 &mydevice의 두번째 객체인 spi통신 인터페이스 신호들을 초기화 및 설정 해주고, 마지막에 Pmod_can을 normal mode로 만들어 메세지를 전송할 수 있게 설정을 해주었다.

2. DemoRun() 함수 분석

```
void DemoRun() {
 CAN_Message RxMessage;
 CAN_RxBuffer target;
 u8 status;
 u8 rx_int_mask;

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

 while (1) {
  do {
   status = CAN_ReadStatus(&myDevice);
   xil_printf("Waiting to receive\r\n");
  } while ((status & CAN_STATUS_RX0IF_MASK) != 0
```

```
    && (status & CAN_STATUS_RX1IF_MASK) != 0);

  switch (status & 0x03) {
  case 0b01:
  case 0b11:
   xil_printf("fetching message from receive buffer 0\r\n");
   target = CAN_Rx0;
   rx_int_mask = CAN_CANINTF_RX0IF_MASK;
   break;
  case 0b10:
   xil_printf("fetching message from receive buffer 1\r\n");
   target = CAN_Rx1;
   rx_int_mask = CAN_CANINTF_RX1IF_MASK;
   break;
  default:
   xil_printf("Error, message not received\r\n");
   continue;
  }

  CAN_ReceiveMessage(&myDevice, &RxMessage, target);  //

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

  xil_printf("received ");
  DemoPrintMessage(RxMessage);

  sleep(1);
 }
}
```

```
typedef struct CAN_Message {
 u16 id;     // 11 bit id
 u32 eid;    // 18 bit extended id
 u8 ide;     // 1 to enable sending extended id
 u8 rtr;     // Remote transmission request bit
 u8 srr;     // Standard Frame Remote Transmit Request
 u8 dlc;     // Data length
 u8 data[8]; // Data buffer
 // Some additional information has not yet been encapsulated here
 // (ex:priority bits), primarily, no TXBxCTRL bits
} CAN_Message;

----------------------------------------code----------------------------------------

XStatus CAN_ReceiveMessage(PmodCAN *InstancePtr, CAN_Message *MessagePtr,
  CAN_RxBuffer target) {
 u8 data[13];
 u8 i;
 u8 read_start_addr;

 switch (target) {
 case CAN_Rx0:
  read_start_addr = CAN_READBUF_RXB0SIDH;
  break;
 case CAN_Rx1:
  read_start_addr = CAN_READBUF_RXB1SIDH;
  break;
 default:
  return XST_FAILURE;
 }

 CAN_ReadRxBuffer(InstancePtr, read_start_addr, data, 13);

 MessagePtr->id = (u16) data[0] << 3;               //  Identifier bits
```

```
 MessagePtr->id |= (data[1] & 0xE0) >> 5;

 MessagePtr->ide = (data[1] & 0x08) >> 3;

 MessagePtr->srr = (data[1] & 0x10) >> 4;

 MessagePtr->eid = (u32) (data[1] & 0x03) << 16;      // Extended Identifier bits
 MessagePtr->eid |= (u32) (data[2] & 0xFF) << 8;
 MessagePtr->eid |= (u32) (data[3] & 0xFF);
 MessagePtr->rtr = (data[4] & 0x40) >> 6;            // Remote Transmission Request bit
 //MessagePtr->dlc = data[4] & 0x0F;
 MessagePtr->dlc &= 0x0;                              // Data Length Code bits
 MessagePtr->dlc |= 0x8;

 // Read only relevant data bytes
 CAN_ReadRxBuffer(InstancePtr, read_start_addr, data, MessagePtr->dlc);

 for (i = 0; i < MessagePtr->dlc; i++)
  MessagePtr->data[i] = data[i + 5];

 return XST_SUCCESS;
}
```

느낀점 : 주제에 맞는 주제를 선정해야한다.

앞으로 할일 : can , spi 공부 그리고 mpu9250

do while 문