ZYBO Z7 10
Petalinux Configuration
/ SPI Control Code

작성자 : Lee DaeRo(skseofh@naver.com)

<Pmod CAN>

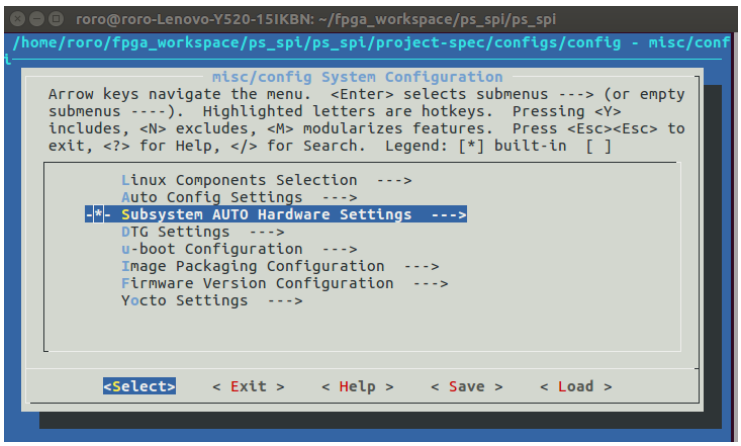
[구현 기능]
- Zybo z7 10에서 TMS570 Launchpad로 CAN 을 이용해 데이터를 전송한다.

[준비물]
- Zybo z7 10
- Pmod CAN
- tms570lc43 launchpad
- CAN transceiver

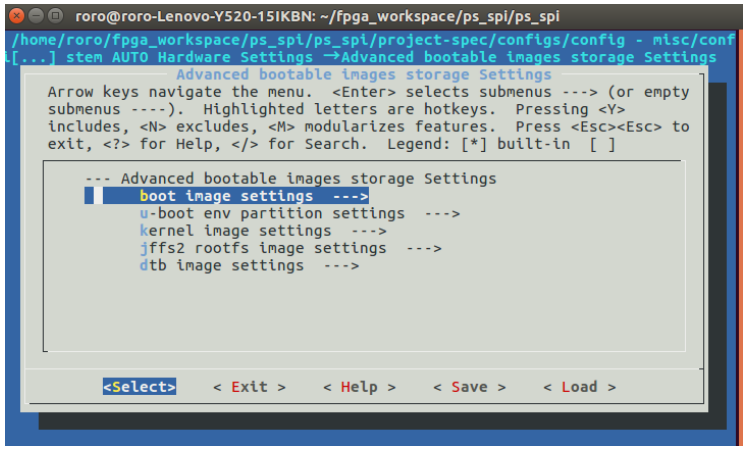
[Petalinux Device File 생성]
1) petalinux project 생성(프로젝트 이름 : ps_spi_i2c)
- 명령어 : petalinux-create --type project --template zynq --name ps_spi_i2c

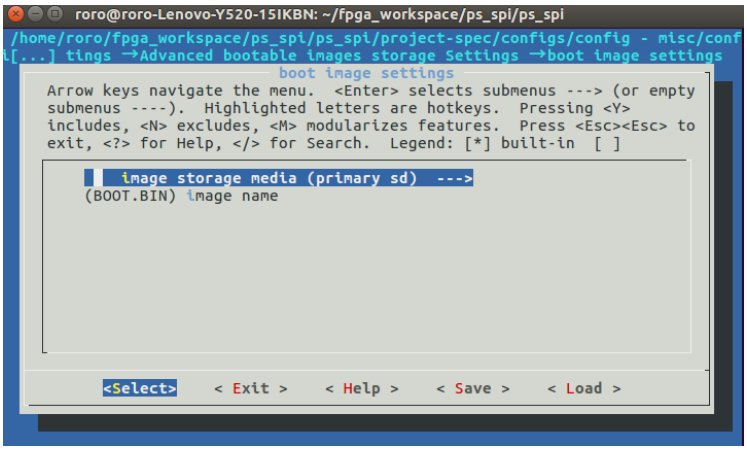
2) Importing Hardware Configuration
- 1)에서 생성된 ps_spi_i2c 디렉터리로 이동
- .hdf 나 .dsa 파일이 들어있는 폴더를 petalinux-config 명령으로 import 시킴
  명령어 : petalinux-config - -get-hw-description=<.hdf 나 .dsa가 들어있는 디렉터리 이름>

```
roro@roro-Lenovo-Y520-15IKBN:~/fpga_workspace/ps_spi/ps_spi$ petalinux-config --
get-hw-description=/home/roro/fpga_workspace/ps_spi/ps_spi.sdk
```

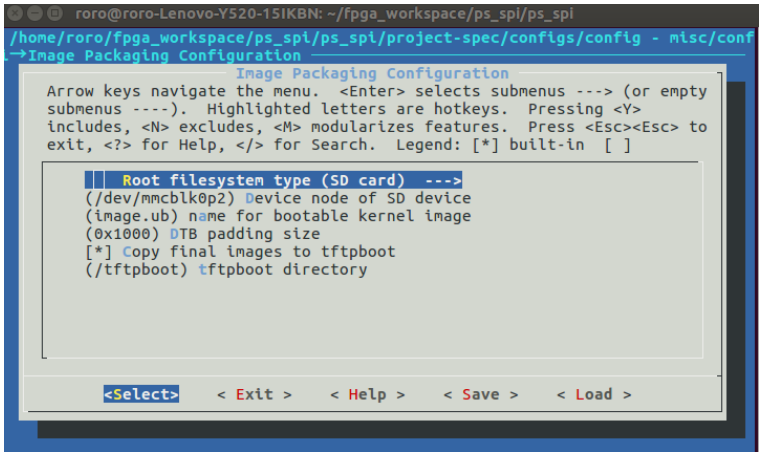


- Subsystem AUTO Hardware Settings → Advanced bootable images storage Settings에서 boot image settings, u-boot env partition settings, kernel image settings, dtb image settings를 모두 primary sd로 변경한다.





- Image Packaging Configuration → Root filesystem type를 SD card로 변경한다.



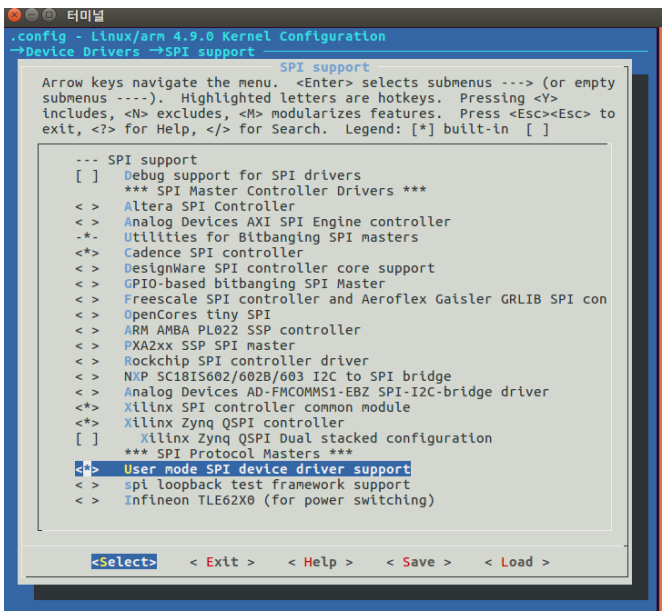
3) petalinux kernel config 시 User mode SPI device driver support 활성화

- 사용자 계층에서 SPI bus에 접근할 수 있도록 drivers/spi/spidev.c을 활성화 시켜주어야 한다.

명령어 : petalinux-config -c kernel

```
roro@roro-Lenovo-Y520-15IKBN:~/fpga_workspace/ps_spi/ps_spi$ petalinux-config -c
 kernel
```

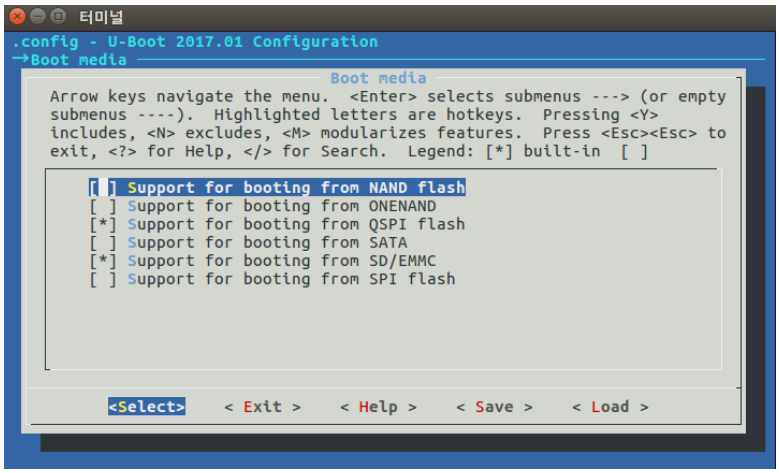
- Device Drivers → SPI support 에서 User mode SPI device driver support를 활성화시킨다.



4) petalinux 부트로더 구성
- 명령어 : petalinux-config -c u-boot

```
roro@roro-Lenovo-Y520-15IKBN:~/fpga_workspace/ps_spi/ps_spi$ petalinux-config -c
 u-boot
```

- Boot media → QSPI flash, SD/EMMC를 활성화 한다.



5) petalinux user app 생성
- 명령어 : petalinux-create -t apps -n <app 이름>-app –enable

```
roro@roro-Lenovo-Y520-15IKBN:~/fpga_workspace/ps_spi/ps_spi$ petalinux-create -t
 apps -n ps-spi-app --enable
```

- <petalinux project 이름>/project-spec/meta-user/recipes-apps에 ps-spi-app 이름의 폴더 생성 확인

6) User app 코드 작성
- ps-spi-app → files → ps-spi-app.c 파일을 수정하여 코드를 작성한다.

7) 디바이스 트리 수정

- <petalinux project 이름>/project-spec/meta-user/recipes-bsp/device-tree/files 폴더에서 system-user.dtsi 파일 수정

```
/include/ "system-conf.dtsi"
/ {
};

&spi0 {
        is-decoded-cs = <0>;
        num-cs = <1>;
        status = "okay";
        spidev@0x00 {
                compatible = "spidev";
                spi-max-frequency = <10000000>;
                reg = <0>;
        };
};
```

8) petalinux build

- 명령어 : petalinux-build

```
roro@roro-Lenovo-Y520-15IKBN:~/fpga_workspace/ps_spi/ps_spi$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image
Loading cache: 100% |#########################################| Time: 0:00:00
Loaded 3258 entries from dependency cache.
Parsing recipes: 100% |#########################################| Time: 0:00:01
Parsing of 2467 .bb files complete (2435 cached, 32 parsed). 3260 targets, 226 s
kipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |#####################################| Time: 0:00:04
Checking sstate mirror object availability: 100% |#############| Time: 0:00:05
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 2457 tasks of which 2375 didn't need to be rerun
and all succeeded.
INFO: Copying Images from deploy to images
NOTE: Failed to copy built images to tftp dir:  /tftpboot
[INFO] successfully built project
```

9) image packaging

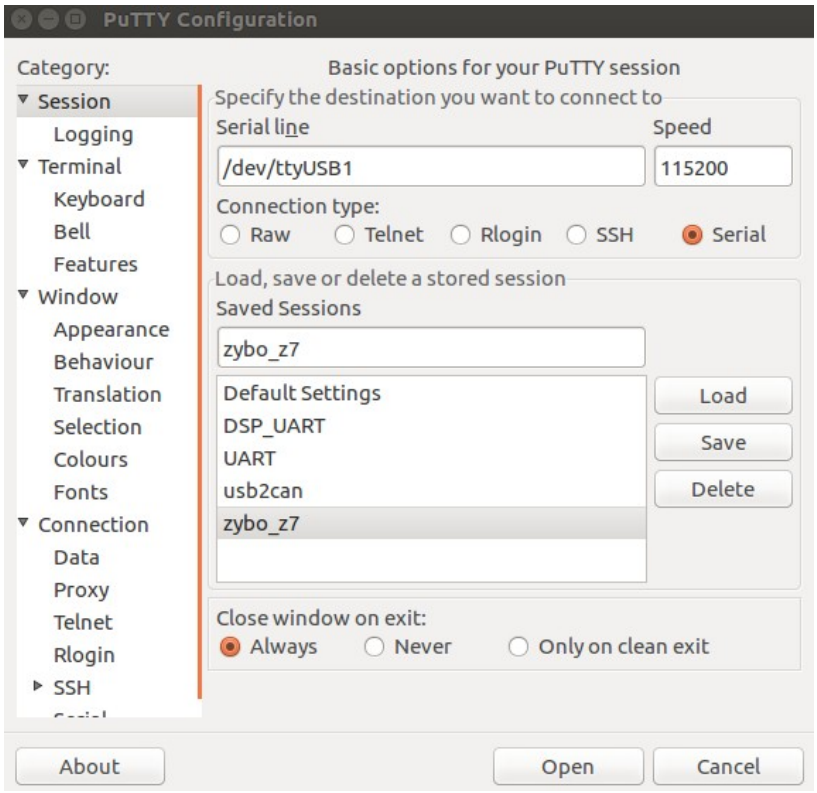
- <petalinux project 이름>/images/linux 폴더로 이동 후 image packaging

- 명령어 : petalinux-package --boot --force --fsbl zynq_fsbl.elf --fpga ps_spi_wrapper.bit –u-boot

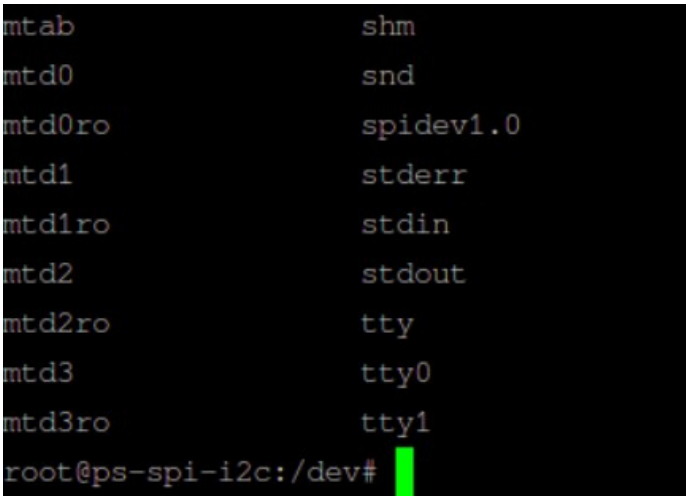
```
roro@roro-Lenovo-Y520-15IKBN:~/fpga_workspace/ps_spi/ps_spi/images/linux$ petali
nux-package --boot --force --fsbl zynq_fsbl.elf --fpga ps_spi_wrapper.bit --u-bo
ot
```

10) SD 카드에서 부팅
- SD 카드의 파티션을 나눈다.(Gparted이용 – BOOT, rootfs)
- BOOT 폴더에는 BOOT.BIN, image.ub, system.dtb 파일을 복사
  rootfs 폴더에는 rootfs.cpio 를 복사
- SD 카드의 rootfs 폴더에서 rootfs.cpio 파일의 pax 명령어로 압축을 해제한다.
  → 명령어 : sudo pax -rvf rootfs.cpio
  (pax가 작동하지 않는 경우에는 sudo apt-get install pax로 설치한다.)

11) ZYBO Z7 10 SD 카드 부트 확인
- SD 카드를 ZYBO 10 SD 카드 Slot에 삽입하고, 전원을 인가한다.
- sudo chmod 666 /dev/ttyUSB1 로 접근권한을 준다.
- terminal 프로그램(ex, putty)를 열어 Serial, /dev/ttyUSB1, 115200으로 설정한다.



- 부팅 완료 되면 root로 로그인을 하고, /dev 에서 spidev1.0이 생성되었는지 확인한다.



[Petalinux User-app Code 작성]

1) 사용자 계층에서 full duplex transfer로 SPI bus 접근 시 ioctl()을 이용함.

```
struct spi_ioc_transfer {
        __u64           tx_buf;
        __u64           rx_buf;

        __u32           len;
        __u32           speed_hz;

        __u16           delay_usecs;
        __u8            bits_per_word;
        __u8            cs_change;
        __u32           pad;
```

```
static void do_msg(int fd, unsigned char * buf, int len)
{
        struct spi_ioc_transfer xfer[2];
        unsigned char                   *bp;
        int                                             status;

        memset(xfer, 0, sizeof xfer);

        if (len > sizeof buf)
                len = sizeof buf;

        xfer[0].tx_buf = (unsigned long)buf;
        xfer[0].len = len;

        xfer[1].rx_buf = (unsigned long) buf;
        xfer[1].len = len;

        status = ioctl(fd, SPI_IOC_MESSAGE(2), xfer);
        if (status < 0) {

                printf("status : %d\n", status);
                perror("SPI_IOC_MESSAGE");
                return;
        }

        printf("response(%2d, %2d): ", len, status);
        for (bp = buf; len; len--)
                printf(" %02x", *bp++);
        printf("\n");

        wait(200);
}
```

- ioctl() 함수를 사용하면 full duplex로 interface 할 수 있다.

→ spi_ioc_tranfer 구조체에서 tx_buf, rx_buf에 버퍼를 설정하고, len변수에는 버퍼 길이 설정

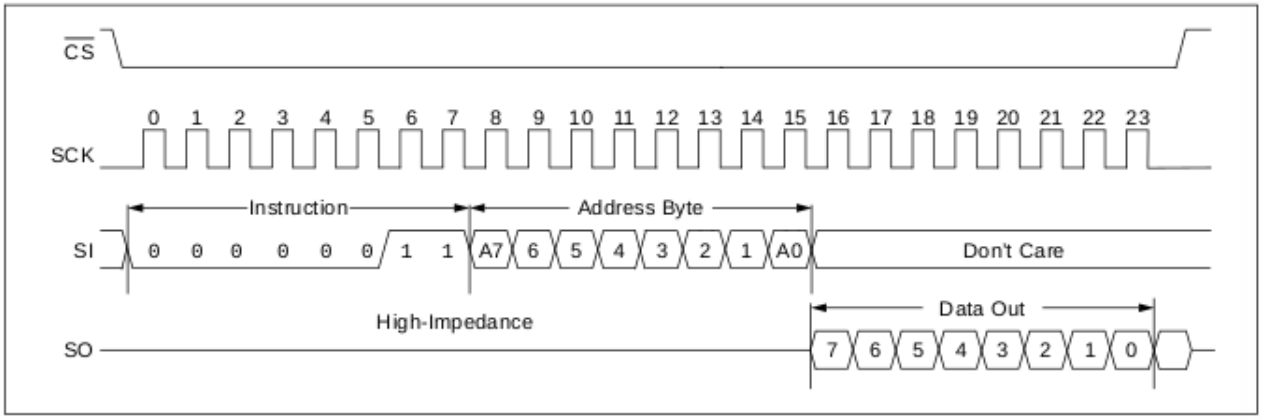
2) Pmod CAN IC 인 mcp25625 명령어 / 레지스터 맵 확인 (mcp25625 datasheet 참조)

< SPI Instruction Set >

TABLE 5-1: SPI INSTRUCTION SET

Instruction Name	Instruction Format	Description
RESET	1100 0000	Resets the internal registers to the default state, sets Configuration mode.
READ	0000 0011	Reads data from the register beginning at the selected address.
READ RX BUFFER	1001 0nm0	When reading a receive buffer, reduces the overhead of a normal READ command by placing the Address Pointer at one of four locations, as indicated by 'nm'.(1)
WRITE	0000 0010	Writes data to the register beginning at the selected address.
LOAD TX BUFFER	0100 0abc	When loading a transmit buffer, reduces the overhead of a normal WRITE command by placing the Address Pointer at one of six locations, as indicated by 'abc'.
RTS (Message Request-to-Send)	1000 0nnn	Instructs the controller to begin the message transmission sequence for any of the transmit buffers. 1000 0nnn Request-to-Send for TXB2 — Request-to-Send for TXBO Request-to-Send for TXB1
READ STATUS	1010 0000	Quick polling command that reads several Status bits for transmit and receive functions.
RX STATUS	1011 0000	Quick polling command that indicates a filter match and message type (standard, extended and/or remote) of the received message.
BIT MODIFY	0000 0101	Allows the user to set or clear individual bits in a particular register.(2)

FIGURE 5-2: READ INSTRUCTION



- 위의 Figure 5-2와 같이 각 Instruction 마다 SPI 명령이 정해져 있음(나머지 명령어는 mcp25625 datasheet 참조)

< Register Map >

- CAN Controller Register Map

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

Note 1: Shaded register locations indicate that these allow the user to manipulate individual bits using the BIT MODIFY command.

- 0x30, 0x40, 0x50에 각각 TX buffer 0, 1, 2가 있다.
- 0x60, 0x70에는 RX buffer 0, 1이 있다.

- Control Register Summary

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

- CANSTAT : CAN 상태 레지스터
  CANCTRL : CAN 컨트롤 레지스터
  CNF3, 2, 1 : Bit Rate 설정 레지스터
  CANINTE : CAN 인터럽트 활성화 레지스터
  CANINTF : CAN 인터럽트 flag 레지스터
  EFLG : 에러 flag 레지스터
  TXBxCTRL : Transmit 버퍼 x 컨트롤 레지스터
  RXBxCTRL : Receive 버퍼 x 컨트롤 레지스터

3) 코드 작성 및 분석

- Library and example code for the Basys MX3 on the Arduino IDE 참조
  https://reference.digilentinc.com/reference/pmod/pmodcan/start

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#define SPI_DEV "/dev/spidev1.0"

#define mcp25625_FAIL    0
#define mcp25625_SUCCESS 1

#define MCP_ALLTXBUSY   2
#define TIMEOUTVALUE    100

#define MCP_SIDH        0
#define MCP_SIDL        1
#define MCP_EID8        2
#define MCP_EID0        3

#define CAN_FAIL            0
#define CAN_OK              1
#define CAN_GETTXBFTIMEOUT  6
#define CAN_SENDMSGTIMEOUT  7

//CANCTRL register values
#define MCP_NORMAL      0x00
#define MCP_SLEEP       0x20
#define MCP_LOOPBACK    0x40
#define MCP_LISTENONLY  0x60
#define MCP_CONFIG      0x80
#define MCP_POWERUP     0xE0
#define MODE_MASK       0xE0

#define MCP_RXB_RX_ANY  0x60
#define MCP_RXB_RX_EXT  0x40
#define MCP_RXB_RX_STD  0x20
#define MCP_RXB_RX_STDEXT   0x00
#define MCP_RXB_RX_MASK 0x60
#define MCP_RXB_BUKT_MASK   (1<<2)

#define MCP_STDEXT  0
#define MCP_STD     1
#define MCP_EXT     2
#define MCP_ANY     3
```

- 헤더 파일 include
- SPI_DEV에 장치 파일 이름 정의 “/dev/spidev1.0”
- 레지스터 관련 정의

```
typedef struct __canMsg
{
    uint16_t    canID;
    uint32_t    canEID;
    uint8_t     canIDE;
    uint8_t     canRTR;
    uint8_t     canSRR;
    uint8_t     canDLC;
    uint8_t     canDATA[8];


}canMsg;
```

- CAN message 구조체 정의
- ID, Extended ID, Extended ID Enable, RTR, SRR, Data Length, Data

- main 함수 -

```
287 canMsg myCanMsg;
288
289 int main(int argc, char **argv)
290 {
291     int fd;
292     uint8_t i, n, res;
293     const char * name = SPI_DEV;
294
295     //CAN message setting
296     uint16_t id = 0x2;
297     uint32_t eid = 0x0;
298     uint8_t ide = 0x0;
299     uint8_t rtr = 0x0;
300     uint8_t dlc = 0x8;
301     uint8_t data[8] = {1, 1, 1, 1, 9, 9, 9, 9};
302
303     char rd_buf[128];
304
305     //device file open
306     if((fd = open(SPI_DEV, O_RDWR))<0)
307     {
308         perror("Open error: ");
309         exit(-1);
310     }
311     printf("Open success\n");
312
313     dumpstat(name, fd);
314
315     //mcp25625 initialize
316     if(mcp25625_init(fd, MCP_ANY, CAN_250KBPS, MCP_20MHZ) == mcp25625_SUCCESS)
317         printf("MCP25625 Initialized Successfully!\n");
318     else
319         printf("Error Initializing MCP25625\n");
320
321     //send CAN message
322     if((res = sendMsgBuf(fd, id, eid, ide, rtr, dlc, data)) !=CAN_OK)
323     {
324         printf("send Msg fail\n");
325     }
326     else
327     {
328         printf("send Msg Success\n");
329     }
330
331     //device file close
332     close(fd);
333
334     return 0;
335 }
```

(1) 보낼 CAN message 정의

- id = 0x02 (standard 형)

  dlc = 0x08

  data[8] = {1, 1, 1, 1, 9, 9, 9, 9};

(2) 장치 파일 open해서 fd를 얻음

(3) mcp25625 initialize

- mcp25625 reset
- bit rate 설정(250kbps)
- CAN buffer reset
- RX 모드 설정
- normal 모드로 변경

(4) CAN message 전송

- CAN message를 버퍼에 setting 후 전송

(5) 장치 파일 close

- mcp25625_init() 함수 -

```
345 bool mcp25625_init(int fd, const uint8_t canIDMode, const uint8_t canSpeed, const uint8_t canClo
    ck)
346 {
347     uint8_t mcpMode, res;
348 
349     //mcp25625 reset
350     mcp25625_reset(fd);
351 
352     //mcp25625 configuration mode setting
353     if(mcp25625_setCANCTRL_Mode(fd, MCP_CONFIG) == mcp25625_SUCCESS)
354         printf("Entering Configuration Mode..\n");
355     else
356     {
357         printf("Fail to Entering Configuration Mode..\n");
358         return mcp25625_FAIL;
359     }
360 
361     //bit rate configuration
362     if(mcp25625_configRate(fd, canSpeed, canClock) == mcp25625_SUCCESS)
363         printf("Bitrate Setting Success!!\n");
364     else
365     {
366         printf("Fail to Setting Bitrate..\n");
367         return mcp25625_FAIL;
368     }
369 
370     //mcp25625 CAN buffer reset
371     mcp25625_initCANBuffers(fd);
372 
373     //mcp25625 RX mode setting
374     mcp25625_modifyRegister(fd, MCP_RXB0CTRL, MCP_RXB_RX_MASK | MCP_RXB_BUKT_MASK,
375                             MCP_RXB_RX_ANY | MCP_RXB_BUKT_MASK);
376     mcp25625_modifyRegister(fd, MCP_RXB1CTRL, MCP_RXB_RX_MASK, MCP_RXB_RX_ANY);
377 
378     //mcp25625 normal mode setting
379     if(mcp25625_setCANCTRL_Mode(fd, MCP_NORMAL) == mcp25625_SUCCESS)
380         printf("Entering to Normal Mode...!\n");
381     else
382     {
383         printf("Fail to Entering Normal Mode..\n");
384         return mcp25625_FAIL;
385     }
386 
387     return mcp25625_SUCCESS;
388 }
```

(1) mcp25625_reset
(2) mcp25625 모드를 configuration 모드로 설정

2.1 CAN Controller Modes of Operation

The CAN controller has five modes of operation:

- Configuration mode
- Normal mode
- Sleep mode
- Listen-Only mode
- Loopback mode

- 4가지 모드 존재
- 설정을 위해서는 Configuration mode 가 되어야 함
- 동작을 위해서는 Normal mode 가 되어야 함.

(3) Bit rate configuration

- 250kbps로 bit rate를 설정하기 위해서 CNF1(0x2A), CNF2(0x29), CNF3(0x28)레지스터를 설정해야 한다.
- mcp25625에는 20MHz 크리스탈이 달려있으므로 그에 맞는 레지스터 값을 설정함
- 설정할 레지스터 값은 #define으로 정의해 놓음

```
225 //  speed 20M
226 #define MCP_20MHz_500kBPS_CFG1 (0x00)
227 #define MCP_20MHz_500kBPS_CFG2 (0xFA)
228 #define MCP_20MHz_500kBPS_CFG3 (0x87)
229
230 #define MCP_20MHz_250kBPS_CFG1 (0x41)
231 #define MCP_20MHz_250kBPS_CFG2 (0xFB)
232 #define MCP_20MHz_250kBPS_CFG3 (0x86)
233
234 #define MCP_20MHz_200kBPS_CFG1 (0x01)
235 #define MCP_20MHz_200kBPS_CFG2 (0xFF)
236 #define MCP_20MHz_200kBPS_CFG3 (0x87)
237
238 #define MCP_20MHz_100kBPS_CFG1 (0x04)
239 #define MCP_20MHz_100kBPS_CFG2 (0xFA)
240 #define MCP_20MHz_100kBPS_CFG3 (0x87)
241
242 #define MCP_20MHz_80kBPS_CFG1 (0x04)
243 #define MCP_20MHz_80kBPS_CFG2 (0xFF)
244 #define MCP_20MHz_80kBPS_CFG3 (0x87)
245
246 #define MCP_20MHz_40kBPS_CFG1 (0x09)
247 #define MCP_20MHz_40kBPS_CFG2 (0xFF)
248 #define MCP_20MHz_40kBPS_CFG3 (0x87)
```

(4) mcp25625 CAN buffer init

- mcp25625의 CAN buffer 레지스터를 reset한다.

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

Note 1: Shaded register locations indicate that these allow the user to manipulate individual bits using the BIT MODIFY command.

(5) RX 모드 설정
- RX0, RX1에 각각 모드를 설정한다.

(6)mcp25625 모드를 Normal 모드로 변경한다.
- CAN 동작을 위해 Normal 모드로 변경

- sendMsgBuf() 함수 -

```
732 //setting message buffer & send message
733 uint8_t sendMsgBuf(int fd, uint16_t id, uint32_t eid, uint8_t ide, uint8_t rtr, uint8_t len, uin
    t8_t * data)
734 {
735     uint8_t res;
736 
737     setMsg(fd, id, eid, ide, rtr, len, data);
738 
739     res = sendMsg(fd);
740 
741     return res;
742 }
```

(1) setMsg()
- CAN message를 버퍼에 설정

(2) sendMsg()
- CAN message 전송

- setMsg() 함수 -

```
744 //set message
745 void setMsg(int fd, uint16_t id, uint32_t eid, uint8_t ide, uint8_t rtr, uint8_t len, uint8_t *
    data)
746 {
747 
748     int i;
749     myCanMsg.canID = id;
750     myCanMsg.canEID = eid;
751     myCanMsg.canIDE = ide;
752     myCanMsg.canRTR = rtr;
753     myCanMsg.canDLC = len;
754 
755     for(i=0; i<len; i++)
756         myCanMsg.canDATA[i] = data[i];
757 
758     printf("canID = %02x\n", myCanMsg.canID);
759     printf("canEID = %02x\n", myCanMsg.canEID);
760     printf("canIDE = %02x\n", myCanMsg.canIDE);
761     printf("canRTR = %02x\n", myCanMsg.canRTR);
762     printf("canDLC = %02x\n", myCanMsg.canDLC);
763 
764     for(i=0; i<len; i++)
765         printf("canDATA[%d] = %02x\n", i, myCanMsg.canDATA[i]);
766 }
```

- 매개변수로 들어온 id, eid, ide, rtr, len, data를 myCanMsg 구조체에 설정한다.

- sendMsg() 함수 -

```
768 //send message
769 uint8_t sendMsg(int fd)
770 {
771
772     uint8_t res, res1, txbuf;
773     uint16_t uTimeOut = 0;
774
775     do{
776         res = mcp25625_getNextFreeTXBuf(fd, &txbuf);
777         printf("selected TXBUF = 0x%02x\n", txbuf);
778         uTimeOut++;
779     }
780     while(res == MCP_ALLTXBUSY && (uTimeOut < TIMEOUTVALUE));
781
782     if(uTimeOut == TIMEOUTVALUE)
783         return CAN_GETTXBFTIMEOUT;
784
785     uTimeOut = 0;
786     mcp25625_write_canMsg(fd, txbuf);
787
788     //mcp25625_modifyRegister(fd, txbuf -1, MCP_TXB_TXREQ_M, MCP_TXB_TXREQ_M);
789
790     do{
791         uTimeOut++;
792         res1 = mcp25625_readRegister(fd, txbuf -1);
793         res1 = res1 & 0x08;
794     }
795     while(res1 && (uTimeOut < TIMEOUTVALUE));
796
797     if(uTimeOut == TIMEOUTVALUE)
798         return CAN_SENDMSGTIMEOUT;
799
800     return CAN_OK;
801
802 }
```

(1) mcp25625_getNextFreeTXBuf()
- 사용가능한 TX 버퍼를 찾는다

(2) mcp25625_write_canMsg()
- CAN message를 전송한다.

- mcp25625_getNextFreeTXBuf() 함수 -

```
689 //get next free TX buffer
690 uint8_t mcp25625_getNextFreeTXBuf(int fd, uint8_t *txbuf)
691 {
692     printf("getNextFreeTXBuf\n");
693 
694     uint8_t res, i, ctrlval;
695     uint8_t ctrlregs[MCP_N_TXBUFFERS] = {MCP_TXB0CTRL, MCP_TXB1CTRL, MCP_TXB2CTRL};
696 
697     res = MCP_ALLTXBUSY;
698     *txbuf = 0x00;
699 
700     for(i=0; i<MCP_N_TXBUFFERS; i++)
701     {
702         ctrlval = mcp25625_readRegister(fd, ctrlregs[i]);
703         if((ctrlval & MCP_TXB_TXREQ_M)==0)
704         {
705             *txbuf = ctrlregs[i] + 1;
706             res = mcp25625_SUCCESS;
707             return res;
708         }
709     }
710     return res;
711 }
```

- TX 버퍼 0, 1, 2 중 free한 버퍼를 선택한다.
→ TXBxCTRL 레지스터에서 TXREQ 비트를 검사하여 0이면 free한 버퍼임

REGISTER 4-1: TXBxCTRL: TRANSMIT BUFFER x CONTROL REGISTER (ADDRESS: 30h, 40h, 50h)

U-0	R-0	R-0	R-0	R/W-0	U-0	R/W-0	R/W-0
—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0
bit 7							bit 0

bit 3 **TXREQ:** Message Transmit Request bit
1 = Buffer is currently pending transmission (MCU sets this bit to request message be transmitted – bit is automatically cleared when the message is sent)
0 = Buffer is not currently pending transmission (MCU can clear this bit to request a message abort)

- mcp25625_write_canMsg() 함수 -

```
804 //write can message
805 void mcp25625_write_canMsg(int fd, const uint8_t buffer_sidh_addr)
806 {
807     uint8_t i, n;
808     uint8_t mcp_addr = buffer_sidh_addr;
809     uint8_t txDLC = myCanMsg.canDLC;
810     unsigned char rts = 0x80;
811     unsigned char txbuf[14];
812     memset(txbuf, 0, sizeof(txbuf));
813
814     mcp25625_setMsg(fd, &rts, txbuf, mcp_addr);
815
816     //request to send
817     do_msg(fd, &rts, 1);
818
819 }
```

(1) 매개변수로 받은 buffer_sidh_addr에는 TX 버퍼 0, 1, 2 중 free한 버퍼의 주소가 들어있음

(2) txbuf[14]를 선언해서 mcp25625_setMsg() 함수에서 CAN message를 mcp25625 레지스터에 load함

(3) RTS(Request to Send) 명령어를 보내서 mcp25625 버퍼에 load한 데이터가 전송될 수 있도록 함.