easyDSP Help

Table of Contents

1. easyDSP 개요 4
2. 제품 종류 6
3. 시작하기 9
4. 업데이트 이력 10
5. 제한 사항 20
6. Pod 구성 21
7. MCU 별 사용법 25
    7.1 C28x 25
        7.1.1 C28x 프로그래밍 25
        7.1.2 C28x 보드 설정 45
        7.1.3 SCI-A 가 아닌 포트 사용 58
        7.1.4 C28x 주의사항 60
    7.2 STM32 61
        7.2.1 STM32 프로그래밍 61
        7.2.2 STM32 하드웨어 설정 69
        7.2.3 STM32 듀얼 코어 74
        7.2.4 STM32 램부팅 관련 설정 80
        7.2.5 STM32 주의사항 84
    7.3 S32 85
        7.3.1 S32K1 + SDK 사용 85
        7.3.2 S32K/S32M + RTD 사용 92
    7.4 AM263x 112
        7.4.1 AM263x 소프트웨어 설정 112
        7.4.2 AM263x 하드웨어 설정 123
    7.5 TM4C 126
    7.6 MSPM0 130
    7.7 PSoC4 137
        7.7.1 PSoC4 하드웨어 설정 137
        7.7.2 PSoC4 소프트웨어 설정 138

7.8 XMC1 149
7.9 XMC4 151
7.10 RA 154
7.10.1 RA 하드웨어 설정 154
7.10.2 RA 소프트웨어 설정 156
7.10.3 RA0 설정 163
7.11 RX 166
7.11.1 RX 하드웨어 설정 166
7.11.2 RX 소프트웨어 설정 167
7.12 TX 176
7.13 TXZ3 180
7.14 LPC 182
7.15 주의 사항 185
8. 메뉴 187
8.1 Project 187
8.2 Edit 192
8.3 MCU 192
8.3.1 공통 192
8.3.2 C28x 194
8.3.3 STM32 200
8.3.4 S32 203
8.3.5 AM263x 205
8.3.6 TM4C 207
8.3.7 MSPM0 209
8.3.8 PSoC4 211
8.3.9 XMC1 212
8.3.10 XMC4 213
8.3.11 RA 214
8.3.12 RX 216
8.3.13 TX, TXZ3 218
8.3.14 LPC 220
8.4 Tools 222

8.5 Window 222
8.6 Help 222
9. 윈도우 223
9.1 Command 223
9.2 Watch 226
9.3 Plot 228
9.4 Chart 232
9.5 Record 234
9.6 Memory 235
9.7 Array 237
9.8 Tree 238
10. 문제 해결 238
10.1 공통 문제 해결 238
10.2 C28x 문제 해결 241
10.3 STM32 문제 해결 247
11. 유용한 팁 249
11.1 DA 컨버터 사용법 249
11.2 기타 251
11.3 자주 묻는 질문들 252
12. Driver 설치방법 253
12.1 설치 방법 253
12.2 Windows 7 에서 254
12.3 제거 방법 258

1. easyDSP 개요

Welcome to easyDSP for real-time MCU monitoring

easyDSP 는 MCU 와 PC 간의 Real-Time 통신 프로그램입니다 .
MCU 통신 포트와의 통신을 통하여 MCU 동작 중 Online 으로 프로그램 변수, 메모리, 레지스터 등을 관찰 및 변경할 수 있습니다. 또한 MCU 부팅 기능을 통하여 램부팅, 플래쉬 롬 라이팅을 할 수 있습니다.
easyDSP 포드 에서 PC 와 MCU 사이를 절연시켜 고신뢰성을 보장합니다. 또한 멀티 운영을 지원하여, 한 개의 PC 에서 여러 개의 easyDSP 를 사용하여 여러 개의 MCU 보드를 제어하는 것이 가능합니다. 지원 가능한 MCU 는 하기 참조하세요.

TI 사 C28x , TM4C , AM263x, MSPM0 시리즈
ST 사 STM32 시리즈
Infineon 사 PSoC4, XMC1, XMC4 시리즈
Renesas 사 RA, RX 시리즈
Toshiba 사 TX, TXZ3 시리즈
NXP 사 LPC1x00, S32K, S32M 시리즈
상세 사항은 여기를 참조하시고, 기타 새로운 MCU 에 대한 지원은 easydsp@gmail.com 으로 문의 바랍니다.

지원하는 OS 는 64bit 윈도우즈 입니다.

국내외 유수의 업체 및 대학에 납품되어 품질 및 성능을 인정 받고 있습니다 .
국내 대학(서울대, 한양대, 숭실대, 성균관대, 아주대, 강원대, 부산대, 제주대, 경남대, 경북대, 영남대, 전북대, 명지대, 건국대, KAIST, 광운대, 한국교통대, 국민대, 목표해양대 등)
국내 기업(삼성전자, 삼성전기, 삼성테크윈, 삼성중공업, LG 전자, LS 산전, 현대중공업, 현대엘리베이터, 온세미컨덕터, 로템, 다사테크, 전자부품연구원, 자동차부품연구원, 한국철도기술연구원, 한국기계연구원, 플라스포, 그린파워, 전력품질기술, 한국전기연구원, 인피니언반도체, 인텍, ADT, 이화전기, LS 메카피온, 한솔테크닉스 ,
한라공조, 두산인프라코어, 두산로보틱스, 효성, 만도, 유니콘, 유니슨, 아주시스템, 이엔테크놀로지,

폴페어, 동양 E&P 등)

국외 기업/대학: Yaskawa, General Motors , Delphi, Raytheon Technologies, Collins Aerospace, Carrier, Grid-bridge, R&D Dynamics , HBL Power System, ADI American Distributors Ltd, Virginia Tech(FEEC@ECE) 등

easyDSP 는 어떠한 보장도 없이 제공됩니다. 제작자가 미리 언급 및 경고하지 않은 경우에도, 어떠한 종류의 손해(직접적, 간접적, 우연적, 필연적, 경제적, 정신적, 신체적)도 보상되지 않습니다.
본 프로그램으로 인한 손해는 전적으로 사용자의 책임이므로 신중히 사용하시기 바랍니다.

기타 자세한 사항은 www.easydsp.com 에서 확인 가능하며 ,
구매, AS, Pod 관련 문의는 hr.oh@egreenpower.com 으로
프로그램 관련 버그, 기능 개선 문의는 easydsp@gmail.com 으로 연락 바랍니다 .
감사합니다.

Welcome to easyDSP

'easyDSP' is a powerful graphical user interface (GUI) for the maintaining, configuring and trouble-shooting of embedded software with strict real-time requirements. The tool automatically extracts the symbol information from the files generated by the cross-compiler and presents the user with windows for the viewing, editing, logging and graphing of those symbols, in real-time, while the target software is executing. easyDSP communicates with the target MCU over a serial communication link, typically SCI. On the target, only a small "remote agent" needs to be called periodically in the background task. Since the remote agent runs on spare processor cycles, it does not interfere with the interrupt driven part of the software. This makes the tool ideal for interfacing with power electronics control software, where the control tasks need to be executed uninterruptedly and with minimal latency. The fact that easyDSP does not depend on JTAG for communicating with the target makes the tool operate reliably in environments with strong EMI and/or high-voltage isolation requirements.

easyDSP is designed for the real-time communication between MCU and an IBM PC or compatible running Windows 64bit.
In case of COFF debugging model with TI C28x, 32bit Windows also supproted.

Supporting :

- TI : C28x, TM4C, AM263x and MSPM0 series
- ST : STM32 series
- Infineon : PSoC4, XCM1 and XMC4 seriels
- Renesas : RA, RX series
- Toshiba : TX and TXZ3 series
- NXP : LPC1x00, S32K and S32M series

The detailed information is available here . For the support of other MCU, please contact easydsp@gmail.com.

easyDSP is not freeware. But it is provided "AS IS" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. In no event shall easyDSP be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if easyDSP has been advised of the possibility of such damage. It is the user's responsibility to check for future updates to the easyDSP and to use the latest version.

For more information, visit www.easydsp.com or
mail to hr.oh@egreenpower.com for purchsing, AS, pod hardware inquiry.
mail to easydsp@gmail.com for bug fix, improvement idea, other technical inquiry.

Thank you.

2. 제품 종류

		
종류 1	종류 2	종류 3

하기의 각 '종류'마다 Pod 가 다르므로, 사용 MCU 및 용도에 따라 적절히 Pod 를 선택 구매하셔야 합니다.

Pod 종류	지원 MCU	MCU 통신 채널	기타
종류 1	TI C28x	SCI	기본 절연형 포드 내부에서 digital isolator 로 절연 구현
종류 2	TI C28x	SCI	광 케이블 절연형 광 케이블 (HFBF1414Z/HFBF2412Z) 채용으로 노이즈 환경에서도 안정적 신호 전송. 장거리 전송도 가능 (전송거리는 200 미터 안에서 주문 형식으로)
종류 3	ST STM32 TI AM263x TI TM4C TI MSPM0 Infineon PSoC4 Infienon XMC1 Infineon XMC4 Renesas RA Renesas RX Toshiba TX, TXZ3 NXP LPC1x, S32	USART 또는 UART 또는 SCI	기본 절연형 포드 내부에서 digital isolator 로 절연 구현

지원되는 MCU 에 대해서는 하기 테이블 참조하세요.

표기법 : [a, b] = a 또는 b. x = 임의의 값.

MCU 종류	파트넘버
TI C28x	TMS320F280[1,2,6,8,9], TMS320F2801[5,6], TMS320F28044, TMS320F281[0,1,2], TMS320F2802[20,30,60,70], TMS320F2802[0,1,2,3,6,7,00],TMS320F2803[0,1,2,3,4,5],TMS320F2805[0,1,2,3,4,5],TMS320F2806[2,3,4,5,6,7,8,9], TMS320F2807[4,5,6], TMS320F28001[32,33,35,37], TMS320F28001[52,53,54,55,56,57],TMS320F28002[1,2,3,4,5],TMS320F28003[3,4,6,7,8,9],TMS320F28004[0,1,5,8,9],TMS320F2823[2,4,5],TMS320F2833[2,3,4,5],TMS320C2834[1,2,3,4,5,6],TMS320F2837[4,5,6,7,8,9]S, TMS320F2837[4,5,6,7,8,9]D, TMS320F2838[4,6,8]S, TMS320F2838[4,6,8]D, TMS320F28P55xS[D,G,J], TMS320F28P65x[S,D][H,K]
TI AM263x	AM263[1,2,4]
TI TM4C	TM4C123[0,1,2,3][C3,D5,E6,H6], TM4C123[6,7][D5,E6,H6], TM4C123[A,B,F,G][E6,H6], TM4C129[0,2]NC, TM4C1294[K,N]C, TM4C1297NC, TM4C1299[K,N]C, TM4C129[C,D]NC, TM4C129E[K,N]C, TM4C129LNC, TM4C129X[K,N]C
TI MSPM0	MSPM0Lxxx[3,4,5,6], MSPM0Gxxx[5,6,7]
ST STM32	STM32C011x[4,6], STM32C031x[4,6], STM32C051x[6,8] NEW , STM32C071x[8,B], STM32C091x[B,C] NEW , STM32C092x[B,C] NEW , STM32F030x[4,6,8,C], STM32F031x[4,6], STM32F031x6, STM32F038x6, STM32F042x[4,6], STM32F048x6, STM32F051x[4,6,8], STM32F058x8, STM32F070x[6,B], STM32F071x[8,B], STM32F072x[8,B], STM32F078xB, STM32F091x[B,C], STM32F098xC, STM32F100x[4,6,8,B,C,D,E], STM32F101x[4,6,8,B,C,D,E,F,G], STM32F102x[4,6,8,B], STM32F103x[4,6,8,B,C,D,E,F,G], STM32F105x[8,B,C], STM32F107x[B,C], STM32F205x[B,C,E,F,G], STM32F207x[C,E,F,G], STM32F215x[E,G], STM32F217x[E,G], STM32F301x[6,8], STM32F302x[6,8,B,C,D,E], STM32F303x[6,8,B,C,D,E], STM32F318x8, STM32F328x8, STM32F334x[4,6,8], STM32F358xC, STM32F373x[8,B,C], STM32F378xC, STM32F398xE, STM32F401x[B,C,D,E], STM32F405x[E,G], STM32F407x[E,G], STM32F410x[8,B], STM32F411x[C,E], STM32F412x[E,G], STM32F413x[G,H], STM32F415xG, STM32F417x[E,G], STM32F423xH, STM32F427x[G,I], STM32F429x[E,G,I], STM32F437x[G,I], STM32F439x[G,I], STM32F446x[C,E], STM32F469x[E,G,I], STM32F479x[G,I], STM32F722x[C,E], STM32F723x[C,E], STM32F730x8, STM32F732xE, STM32F733xE, STM32F745x[E,G], STM32F746x[E,G], STM32F750x8, STM32F756xG, STM32F765x[G,I], STM32F767x[G,I], STM32F769x[G,I], STM32F77[7,8,9]xI, STM32G030x[6,8], STM32G031x[4,6,8], STM32G041x[6,8], STM32G050x[6,8], STM32G051x[6,8], STM32G061x[6,8], STM32G070xB, STM32G071x[8,B], STM32G081xB, STM32G0B0xE, STM32G0B1x[B,C,E], STM32G0C1x[C,E], STM32G431x[6,8,B], STM32G441xB, STM32G473x[B,C,E],

	STM32G474x[B,C,E], STM32G483xE, STM32G484xE, STM32G491x[C,E], STM32G4A1xE, STM32H503xB, STM32H523x[C,E], STM32H533xE, STM32H562xG, STM32H562xI, STM32H563xG, STM32H563xI, STM32H573xI, STM32H723x[E,G], STM32H725x[E,G], STM32H730xB, STM32H733xG, STM32H735xG, STM32H742x[G,I], STM32H743x[G,I], STM32H745x[G,I], STM32H747x[G,I], STM32H750xB, STM32H753xI, STM32H755xI, STM32H757xI, STM32H7A3x[G,I], STM32H7B0xB, STM32H7B3xI, STM32L010x[4,6,8,B], STM32L011x[3,4], STM32L021x4, STM32L031x[4,6], STM32L041x6, STM32L051x[6,8], STM32L052x[6,8], STM32L053x[6,8], STM32L06[2,3]x8, STM32L07[1,2,3]x[8,B,Z], STM32L08[1,2]x[B,Z], STM32L083x[8,B,Z], STM32L100x[6,8,B,C], STM32L151x[6,8,B,C,E], STM32L152x[6,8,B,C,D,E], STM32L162x[C,D,E], STM32L100x[8,B]-A, STM32L151x[6,8,B,C]-A, STM32L152x[6,8,B,C]-A, STM32L162xC-A, STM32L15[1,2]xD-X, STM32L162xD-X, STM32L412x[8,B], STM32L422xB, STM32L431x[B,C], STM32L432x[B,C], STM32L433x[B,C], STM32L442xC, STM32L443xC, STM32L451x[C,E], STM32L452x[C,E], STM32L462xE, STM32L471x[E,G], STM32L475x[C,E,G], STM32L476x[C,E,G], STM32L486xG, STM32L496AE, STM32L496x[E,G], STM32L4A6xG, STM32L4P5x[E,G], STM32L4Q5xG, STM32L4R5x[G,I], STM32L4R7xI, STM32L4R9x[G,I], STM32L4S[5,7,9]xI, STM32L552x[C,E], STM32L562xE, STM32U031x[4,6,8], STM32U073x[8,C], STM32U083xC, STM32U375x[E,G] NEW , STM32U385xG NEW , STM32U535x[B,C,E], STM32U545xE, STM32U575x[G,I], STM32U585xI, STM32U595x[I,J], STM32U599x[I,J], STM32U5A5x[I,J], STM32U5A9xJ, STM32U5F[7,9]xJ, STM32U5G[7,9]xJ, STM32WB[10,15]xC, STM32WB30xE, STM32WB35x[C,E], STM32WB50xG, STM32WB55x[C,E,G,Y], STM32WB05xZ , STM32WB06xC , STM32WB07xC , STM32WB09xE, STM32WBA50xG NEW , STM32WBA52x[E,G], STM32WBA5[4,5]x[E,G], STM32WL33x[8,B,C], STM32WL5[4,5]xC, STM32WLE[4,5]x[8,B,C]
Infineon PSoC4	CY8C402[4,5], CY8C404[5,6], CY8C412[4,5], CY8C4126xxx-S42x, CY8C4126xxx-S43x, CY8C4126xxx-S44x, CY8C4126xxx-S45x, CY8C4126xxx-Mxxx, CY8C4127xxx-Sxxx, CY8C4127xxx-Mxxx, CY8C4127xxx-BLxxx, CY8C4128xxx-Sxxx, CY8C4128xxx-BLxxx, CY8C414[5,6,7,8], CY8C424[4,5], CY8C4246xxx-DSxxx, CY8C4246xxx-Mxxx, CY8C4246xxx-Lxxx, CY8C4247xxx-Mxxx, CY8C4247xxx-Lxxx, CY8C4247xxx-BLxxx, CY8C4248xxx-Lxxx, CY8C4248xxx-BLxxx, CY8C454[6,7,8], CY8C472[4,5], CY8C474[4,5]
Infienon XMC1	XMC1100-xxxxx0[008,016,032,064], XMC120x-xxxxx0[016,032,064,128,200], XMC130x-xxxxx0[016,032,064,128,200], XMC140x-xxxxx0[032,064,128,200],
Infineon XMC4	XMC410x-xxxx[64,128], XMC4200-xxxx256, XMC4300-xxxxx256, XMC440x-xxxxx[256,512], XMC450x-xxxxx[512,768,1024], XMC4700-xxxxx[1536,2048], XMC4800-xxxxx[1024,1536,2048]

Renesas RX	R5F5110[1,3,4,5,H,J], R5F5111[1,3,4,5,6,7,8,J], R5F5113[5,6,7,8], R5F5130[3,5,6,7,8], R5F513T[3,5], R5F5140[3,5,6], R5F5230[5,6], R5F5231[5,6,7,8], R5F523E[5,6], R5F523T[3,5], R5F523W[7,8], R5F524T[8,A,B,C,E], R5F524U[B,C,E], R5F526T[8,9,A,B,F], R5F5651[4,7,9,C,E], R5F565N[4,7,9,C,E,D,N], R5F5660[4,9], R5F566N[D,N], R5F566T[A,E,F,K], R5F5671[9,C,E], R5F571M[F,G,J,L], R5F572M[D,N], R5F572N[D,N], R5F572T[F,K]
Renesas RA	R7FA0E1x[5,7] NEW , R7FA2A1xB, R7FA2A2xD, R7FA2E1x[5,7,9], R7FA2E2x[3,5,7], R7FA2E3x[5,7], R7FA2L1x[9,B],R7FA4E1x[B,D], R7FA4E2x9, R7FA4L1x[B,D] NEW, R7FA4M1AB, R7FA4M2x[B,C,D], R7FA4M3x[D,E,F], R7FA4T1x[9,B], R7FA4W1xD, R7FA6E1x[D,F], R7FA6E2x[9,B],R7FA6M1xD, R7FA6M2x[D,F], R7FA6M3x[F,H], R7FA6M4x[D,E,F], R7FA6M5x[F,G,H], R7FA6T1x[B,D], R7FA6T2x[B,D], R7FA6T3xB, R7FA8D1x[F,H], R7FA8M1x[F,H], R7FA8T1x[F,H], R7FA8E1xF, R7FA8E2xF
Toshiba TX, TXZ3	TMPM03[6,7]FW, TMPM06[1,6,7,8]FW, TMPM330F[D,W,Y], TMPM332FW, TMPM333F[D,W,Y], TMPM341F[D,Y], TMPM365FY, TMPM366FD, TMPM367FD, TMPM368FD, TMPM369FD,TMPM370FY, TMPM372FW, TMPM373FW, TMPM374FW, TMPM375FS, TMPM376FD, TMPM37AFS,TMPM380F[W,Y], TMPM381FW, TMPM383F[S,W], TMPM384FD,TMPM3U0FS, TMPM3U6F[W,Y], TMPM3V4F[S,W], TMPM3V6FW, TMPM440F[10,E], TMPM461F[10,15], TMPM462F[10,15], TMPM46BF10,TMPM3H0F[M,S], TMPM3H1F[P,S,W,U], TMPM3H2F[S,U,W], TMPM3H3F[S,U,W], TMPM3H4F[S,U,W], TMPM3H5F[S,U,W], TMPM3H6F[S,U,W], TMPM3HLF[D,Y,Z], TMPM3HMF[D,Y,Z], TMPM3HNF[D,Y,Z], TMPM3HPF[D,Y,Z], TMPM3HQF[D,Y,Z]
NXP S32 NEW	S32K11[6,8], S32K14[2,4,6,8], S32K31[0,1,2,4], S32K34[1,2,4,8], S32M24[1,2,3,4], S32M27[4,6]
NXP LPC1xxx	LPC13x[1,2,3], LPC131[5,6,7], LPC134[5,6,7], LPC15x[7,8,9], LPC175[1,2,4,6,8,9], LPC176[3,4,5,6,7,8,9], LPC177[3,4,6,7,8], LPC178[5,6,7,8], LPC181[2,3,5,7], LPC182[2,3,5,7], LPC183[3,7], LPC185[3,7], LPC18S[3,5]7

새로운 MCU 지원이 필요한 경우, easyDSP@gmail.com 으로 문의 바랍니다.

3. 시작하기

순서	작업	설명
1	easyDSP 와 MCU 보드 연결	MCU 와 easyDSP 간의 하드웨어 연결을 위한 작업입니다. 상세사항은 본 도움말의 'MCU 별 사용법' 참조하세요.
2	사용자 프로그램 수정	먼저 easyDSP 가 제공하는 소스 파일 및 헤더 파일을 사용자 프로젝트에 포함시킵니다. easyDSP 가 설치된 폴더안의 Source 폴더에서 해당 파일을 찾으실 수 있습니다.

		이후 헤더 파일에서 #define 전처리기 변수를 시스템에 맞게 설정합니다. MCU 종류에 따라 필요하지 않을 수 있습니다. main.c 에서 헤더파일을 include 하고 easyDSP 통신 설정을 위한 함수를 호출합니다. 상세사항은 본 도움말의 'MCU 별 사용법' 참조하세요.
3	easyDSP 프로젝트 생성	본 도움말의 '메뉴 > Project'를 참조하여 easyDSP 프로젝트를 생성합니다.
4	MCU 부팅	램부팅 또는 플래시 프로그래밍으로, MCU 를 부팅합니다. 본 도움말의 '메뉴>MCU' 참조하세요.
5	easyDSP 를 사용한 MCU 모니터링	easyDSP 각종 윈도우를 통해 MCU 변수를 모니터링합니다.
6	사용자 프로그램 변경	필요시 IDE 환경에서 사용자 프로그램을 업데이트 합니다.
7	MCU 부팅	순서 4 를 수행하여 변경된 사용자 프로그램으로 MCU 를 부팅합니다.
8	easyDSP 를 사용한 MCU 모니터링	순서 5 을 수행합니다.

4. 업데이트 이력

버전	MCU	항목
ver 11.4 Apr/2025	공통	- 일부 MCU 의 플래시 프로그래밍에서 지울 섹터(페이지) 선택을 비활성화하는 기능 도입 (Freeze 체크 박스) **수정된 버그 :** Y 축 범위를 자동으로 설정할 때, 여러 Plot 창에 동일한 Y 축 범위가 적용됨
	TI C28x	- Gen.3 MCU 경우,섹션 주소의 얼라인먼트가 맞지 않을 때에도 플래시 프로그래밍 지원 - 일부 MCU 플래시 프로그래밍 대화상자 변경 (F2837xS, F2807x, F28002x, F28003x, F28004x) **수정된 버그 :** - Gen.3 MCU : 섹션 크기가 0xFFFF 보다 클 때 섹션 주소 얼라인먼트 실패로 플래시 프로그래밍 못할 경우가 있음 - F28Px, F280001x 의 플래시 대화상자에서 특정 경우 사용자 프로그램에 사용된 sector 를 잘못 인식
	ST STM32	- 전체 플래시 지우는 기능 추가 (플래시 대화 상자에 Erase chip 버튼) - STM32H72x, STM32H73x 의 내장 부트로더의 특정 버전에서 플래시 프로그래밍시 주기적으로 32B 의 0xFF 가 잘못 프로그래밍되는 오류 개선 - STM32U375x[E,G], STM32U385xG (소스파일 easyStm32LL_v11.4.c 사용 필요)

		- STM32WBA50xG, STM32C051x[6,8], STM32C091x[B,C], STM32C092x[B,C] 지원 - STM32L471xE, STM32L475x[C,E], STM32L476x[C,E], STM32L496xE 에 관해 single bank, dual bank 를 구분하여 지원 **수정된 버그 :** - STM32U5, STM32L5, STM32H7, STM32G0 에서 SWAP_BANK bit = 1 인 dual bank 경우 플래시 프로그래밍 오동작 - STM32H7 dual core 또는 STM32WL3x 사용시 소스파일 easyStm32LL v11.3.c 에서 컴파일 오류 발생 - STM32F76[5,7,9]xI, STM32F77[7,8,9]xI MCU 의 dual bank 구성시 페이지 범위 오류 - STM32WL33x 플래시 프로그래밍 지원 안됨
	NXP S32	- NXP MCU S32K/S32M 시리즈 지원 : S32K11[6,8], S32K14[2,4,6,8], S32K31[0,1,2,4], S32K34[1,2,4,8], S32M24[1,2,3,4], S32M27[4,6]
	Renesas RA	- R7FA4L1x[B,D], R7FA0E1x[5,7] 지원 (소스파일 easyRA v11.4.c 사용 필요)
ver 11.3 Jan/2025	공통	DWARF5 지원 개선
	ST STM32	- STM32WB0[5,6,7,9] 지원(소스파일 easyStm32LL_v11.3.c 사용 필요) - STM32H523x[C,E], STM32H533xE 지원 - STM32C071x[8,B] 지원 **수정된 버그 :** STM32WB09xE 지원 오류
	Renesas RA	- R7FA8E1xF,R7FA8E2xF 지원
	Renesas RX	**수정된 버그** : 8 바이트 변수,포인터 변수 모니터링 오류 (ver 11.1 부터의 버그)
ver 11.2 May/2024	TI C28x	- TMS320F28P55xS 지원 (소스파일 easy28x_bitfield_v11.2.c 사용 필요) **수정된 버그 :** F28P65xDH 플래시 영역 뱅크 4 프로그래밍 오동작
	ST STM32	- STM32U0 지원 (소스파일 easyStm32LL_v11.2.c 사용 필요) - STM32U5A5xI, STM32U5F7xJ, STM32U5F9xJ, STM32U5G7xJ, STM32U5G9xJ 지원 - STM32WB09xE, STM32WBA54x[E,G], STM32WBA55x[E,G], STM32WL33x[8,B,C] 지원
	Renesas RA	- RA8T1, RA2A2 지원 **수정된 버그 :** RA8D1 선택시 프로젝트 생성 안됨
	Renesas RX	- RX23E-B 지원

ver 11.1 Jan/2024	공통	- 플래시 영역에 존재하는 변수의 쓰기 금지 **수정된 버그 :** Plot 윈도우에서 Total plot period 를 71582 분 이상으로 설정시 오동작
	ST STM32	- STM32U535, STM32U545, STM32U595, STM32U599, STM32U5A5, STM32U5A9 시리즈 지원
	Renesas RA	- RA2E3, RA4E2, RA4T1, RA6E2, RA6T3, RA8D1 , RA8M1 지원 (easyRA_v11.1.c, easyRA_v11.1.h 파일 사용 필요) **수정된 버그 :** RA6M5 계열 플래시용량이 1.5MB 보다 같거나 큰 MCU 경우, 플래시 프로그래밍 지원 불가
	Renesas RX	- RX26T 지원
	NXP LPC1xxx	- LPC1500 플래시 프로그래밍 지원 - LPC1300, LPC1700, LPC1800 (플래시 내장 타입) 시리즈 지원 **수정된 버그 :** 공용체 배열 변수의 주소 인식 오류
	TI AM2x	- 램부팅 및 플래시 프로그래밍관련 변경 (앱 이미지 파일 변경 가능, SBL 보드레이트 변경 가능, SBL 이미지 파일은 easyDSP 가 제공하지 않음) **수정된 버그 :** easyDSP 설치 폴더 안에 util 폴더에 MulticoreImageGen.exe 파일 없음. v10.8 - v11 의 버그.
ver 11.0 9/2023	TI C28x	- TMS320F28P65x 지원 (소스파일 v11 필요) - F2837xD, F2838xD CPU2 램부팅시 메모리 관리 방식 개선 (소스파일 v11 필요) **수정된 버그 :** F2838xD CPU1 에서 DriverLib 소스파일 사용시 CPU1, CPU2, CM 을 모두 플래시부팅할 경우, CM 플래시 부팅 실패 (소스파일 easy28x_driverlib_v11.c 필요)
	TI MSPM0	- MSPM0 계열 MCU 지원
ver 10.9 6/2023	ST STM32	- STM32H5, STM32WBA 시리즈 지원 (소스 파일 easyStm32LL v10.9.c 필요) 수정된 버그 : STM32H7, STM32L0, STM32L1, STM32L5, STM32U5 에 대해서 특정 경우 플래시 프로그래밍 오동작으로 Verify 시 에러 메시지 발생
	NXP LPC1500	NXP 1500 시리즈 지원 (플래시 프로그래밍 미지원)
ver 10.8 4/2023	공통	- 다차원 배열 경우 10 차원 배열까지 지원 (이전 버전은 4 차원 배열까지만 지원) - Array 창 : 선택된 셀을 클립보드로 복사할 때, 빈셀이 있을 경우 먼저 셀의 값을 채움
	TI C28x	- TMS320F280015x 지원 (새로운 소스 파일 easy28x_bitfield_v10.8.c 또는 easy28x_driverlib_v10.8.c 필요)

		수정된 버그 : 특정 경우 TMS320F280013x 플래시 프로그래밍 오류 발생
	ST STM32	- STM32 C0 계열 지원 (소스 파일 easyStm32LL v10.8.c 필요)
	Renesas RX	- 르네사스 MCU RX 계열 지원
ver 10.7 1/2023	공통	- Watch 창에서 변수 항목을 위/아래로 이동 가능 - Watch, Memory, Array, Tree 창: 값 변경시 해당 셀을 노란 배경색으로 표시 (프로젝트 설정에서 선택 가능)
	TI AM2x	- AM263x 지원
	TI TM4C	- TM4C123x, TM4C129x 지원
ver 10.6 11/2022	TI C28x	- TMS320F280013x 지원 (새로운 소스 파일 easy28x_bitfield_v10.6.c 또는 easy28x_driverlib_v10.6.c 필요) - 플래시 관련 동작을 수행하기 위해 easyDSP 프로그램을 관리자 모드로 실행할 필요가 없음 **수정된 버그 :** 멀티 코어의 경우 CPU2 부터 변수 정보 읽어 들이지 못함 (v10.5.1 만의 버그)
ver 10.5.1 11/2022	TI C28x	**수정된 버그 :** 특정 경우플래시 대화상자 진입시 오류 메시지 "The variables in flash API wrapper are not fully recognized!" 발생 이후 플래시 작업 하지 못함 (v10.3 이후의 버그)
ver 10.5 11/2022	공통	- 이전 형식 메모리 윈도우 미지원 - 메모리 윈도우 : 주소 입력시 변수 주소 형식을 (예 : &var) 사용한 경우, 변경된 출력 파일로 새로 MCU 를 부팅한 경우 해당 변수 주소가 변경되었다면, 메모리 윈도우의 주소도 자동 변경. **수정된 버그 :** - 커맨드 윈도우 : 구조체/공용체/비트필드 변수의 자동 찾기 기능 동작 안함
	ST STM32	- easyDSP 소스파일이 easyStm32LL_v10.5.c 로 업그레이드 되어 하기 기능을 지원합니다. 1. STM32G0x : 램부팅/ 플래시프로그래머 대화상자에서 부트로더 진입 개선 2. STM32H7 dual core (STM32H745x, STM32H747x, STM32H755x, STM32H757x) : 데이타 캐시 사용 가능 3. USART 에 FIFO 기능이 있는 MCU 에서, easyDSP 통신을 빠르게 하기 위해, FIFO 사용 가능

	Toshiba TXZ3	- Toshiba 사 TXZ3 MCU 지원
ver 10.4 5/2022	공통	**수정된 버그 :** - ARM 계열 MCU 에서 DWARF4 또는 DWARF5 디버깅 정보 포맷 사용시, 특정 경우 비트필드 변수의 주소 오류 발생.
	TI C28x	**수정된 버그 :** - 2838x 경우 CPU1, CPU2, CM 을 모두 사용하지 않을 경우, 플래시 작업 불가능. v10.3 및 v10.3.1 만의 버그
	Toshiba TX	- Toshiba 사 TX MCU 지원
ver 10.3.1 4/2022	TI C28x	**수정된 버그 :** - 2838x 경우 CPU1, CPU2, CM 을 모두 사용하지 않을 경우, easyDSP 프로젝트가 열리지 않음. v10.3 만의 버그.
ver 10.3 4/2022	공통	- 와치 윈도우에서 비트필드 멤버 변수의 주소에 비트 정보 포함 (예를 들어 0x1234@bit1-2 형식) - 비트필드 멤버 변수의 표시 진수 절환 가능 - 차트 윈도우에서 1 차원 배열 변수만 사용하도록 변경 - 트리 윈도우에서 마우스 우클릭으로 변수값 표시 진수 절환 (dec -> hex -> bin -> dec) - 메모리 윈도우에서 주소 입력 형식 다양화 및 주석 가능 - 변수 정보 처리 속도 향상 - 드라이버 파일 업데이트 CDM212364_Setup.exe **수정된 버그 :** - 익명(Anonymous)의 구조체/공용체 변수 멤버 표시 오류 - 익명(Anonymous)의 비트필드 구조체 멤버 변수 표시 오류 - 4 바이트보다 큰 크기의 비트필드 멤버 표시 오류
	TI C28x	- 2837xD, 2838xS/D 멀티 코어를 2 개 이상의 프로젝트로 사용시, CPU1 프로젝트에서 out 파일이 업데이트될 때 CPU2/CM 프로젝트에서 메시지 송출 - COFF 디버깅 모델에 대해 지원되었던 32 비트 윈도우즈 지원 중단 - 레지스터 윈도우 지원 중단 - 플래시 대화상자에서 flash API speed [bps] 속도 변경 기능 (하기 v10.1 내용 참조)의 동작 안정성 향상 **수정된 버그 :** - 2837xD/2838xS/D 멀티코어 사용시, CPU1 프로젝트에서 플래시 대화상자 진입 이후, CPU2/CM out 파일이 업데이트 되고 플래시 프로그래밍에 사용될 때 특정

		경우 bin 파일이 없다는 에러 발생 - 2837xD/2838xS/D 멀티코어 사용시 CPU2/CM 프로젝트 없이 CPU1 프로젝트만 사용하여 CPU2/CM 을 램부팅/플래시라이팅할 때, 특정 경우, 현재 out 파일이 아닌 이전 out 파일로 램부팅/플래시라이팅 진행 - 2837xD 에서 coff 디버깅 모델 사용시 CPU2 프로그램이 갱신되지 않음.
	ST STM32	- 램부팅, 플래시 프로그래밍시 IDE 에서 생성한 hex 파일만을 사용. 기존 버전에서 easyDSP 가 hex 파일을 생성하는 옵션을 사용하셨다면 주의 바랍니다. 이제는 IDE 에서 매 컴파일시 hex 파일을 생성하도록 옵션을 설정해야 합니다. **수정된 버그 :** IDE 가 생성한 hex 파일을 사용할 경우, 플래시 대화상자에서, 변경된 유저 프로그램 사용 여부 선택과 상관없이, 무조건 변경된 유저 프로그램이 사용됨.
	Infineon PSoC4	**수정된 버그 :** 플래시 대화상자에서, 변경된 유저 프로그램 사용 여부 선택과 상관없이, 무조건 변경된 유저 프로그램이 사용됨.
	Infineon XMC4	**수정된 버그 :** 플래시 대화상자에서, 변경된 유저 프로그램 사용 여부 선택과 상관없이, 무조건 변경된 유저 프로그램이 사용됨.
	Renesas RA	Renesas 사 MCU RA 시리즈 지원
	Infineon XMC1	Infineon 사 XMC1 MCU 지원 (모니터링만 지원. 프로그래밍 미지원)
ver 10.2 1/2022	TI C28x	**수정된 버그 :** F2837xS : 플래시 대화상자가 열리지 않음 (v10.1 의 버그)
	Infineon PSoC4	- Infineon 사 PSoC4 MCU 지원
	Infineon XMC4	- Infineon 사 XMC4 MCU 지원
ver 10.1 11/2021	공통	- 새로운 형식의 메모리 창 (상세 확인) **수정된 버그 :** - char 또는 unsigned char 타입 변수가 아니어도 문자(ex, 'A')가 값으로 입력 - Array 윈도우에서 char 또는 unsigned char 타입 변수에 문자(ex, 'A') 입력 안됨. - 부동소수점 변수 (float, double, long double) 포인터 변수에 부동소수점 값 입력 가능
	TI C28x	- F28003x : 지원 시작 (easyDSP 제공 소스파일 버전 v10.1 사용 필요) 주의 사항) 요구되는 최소 사양으로 CCSv11 과 컴파일러 버전 21.6.0.LTS - F2837xS/F2837xD/F28004x/F28002x/F2838xD/F2838xS/ F2802x/F2802x0/F2803x/F2805x/F2806x/F2807x : 플래쉬 동작 속도 최대 2 배 개선

		Flash API speed [bps] 115200 본 메뉴로 통신 속도를 조절하여 시간 단축 가능. 단, 특정 bps 는 동작하지 않을 수 있음. - F2807x/F2837xS/F2837xD 내부 클럭 소스 지원 (외부 클럭 소스 없이 사용 가능) - F2802x Rev.0 지원 중지 - F2838xS/D CM : 램부팅의 'Enables fast verifying' 기능 비활성화 - 멀티 코어 2837xD 및 2838xS/D : CPU1 용 easyDSP 프로젝트에서 램부팅 또는 플래시 프로그래밍 수행시에, CPU2 또는 CM 용 easyDSP 프로젝트에서 통신을 잠시 멈춤. 단, CPU1, CPU2 또는 CM 용 easyDSP 프로젝트가 동일 PC 에서 동작할 경우에만. - easyDSP DriverLib 소스파일 개선 (easy28x_driverlib_v10.1.c) : 28003x 지원, Gen3 MCU 32 비트 어드레스 영역 지원, C2000Ware_4_00_00_00 에서 변경된 핀 Mux 이름 지원 - easyDSP DriverLib 소스파일 개선 (easy28x_cm_driverlib_v10.1.c) : EtherCAT RAM 및 ECC 메모리 영역 억세스 가능, Hard Fault 방지를 위한 메모리 번지 검증 기능 추가 - easyDSP BitField 소스파일 개선 (easy28x_bitfield_ v10.1 .c) : 28003x 지원, Gen3 MCU 32 비트 어드레스 영역 지원 **수정된 버그 :** - 공용체/구조체 변수 인식 오류 (v10 의 버그) - 메모리창에서 Gen3 MCU 의 TI OTP 메모리 영역을 읽을 때 시스템 에러 발생 - F2838xS/D CM : 특정 경우 램부팅 verify 실패
	ST STM32	- HAL 기반 easyDSP 통신 소스 지원 중단 (LL 기반 대비 많은 리소스 소모로 인해) - LL 기반 easyDSP 통신 소스 개선 (address alignment check) : easyStm32LL_v10.1.c 통신 소스 코드 사용 필요 - STM32WB10xC, WB15xC 지원 - STM32U5 시리즈 지원 **수정된 버그 :** - 플래시 페이지 크기가 128 바이트인 일부 MCU 에서 사용된 페이지를 판별하지 못함
ver 10 5/2021	공통	**수정된 버그 :** 트리윈도우 구조체/공용체 변수 리스트에서 구조체/ 공용체가 아닌 변수가 등록됨
	TI C28x	- F2837xS/F2837xD/F2807x 부팅 시 오토 보딩 알고리즘 개선 - C++의 class 타입 변수 지원

		- 플래시롬 대화상자 관련 개선 (C2834x 제외) 1. 플래시롬 대화상자에서 "Erase > Program..."과 같이 연속 동작시에는, 프로그램될 플래시 섹터가 지워질 섹터에 다 포함되었는지를 확인하는 기능 추가 2. 한번 클릭으로 모든 동작을 수행하는 버튼 추가 3. out 파일 갱신시 사용자 수락 여부를 플래시롬 대화상자 진입시 확인하지 않고, 추후 플래쉬롬 동작(Program, verify, select used, select not used)시 확인 **수정된 버그 :** - F2837xD, F2838xS/D 에서 out 파일 갱신되었을 때, 사용자의 수락 여부와 상관없이 CPU2, CM 의 경우 갱신된 out 파일을 프로그램함 - F2837xD, F2838xS/D 의 플래시롬 또는 램부팅 대화상자 진입 이후 CPU2 또는 CM 의 out 파일이 갱신되었을 때, CPU2 또는 CM 과 통신하는 easyDSP 프로그램이 비활성화 상태라면 CPU2, CM 에는 갱신된 out 파일이 프로그램되지 않음
	ST STM32	- ST 사 STM32 MCU 시리즈 지원 (전용 easyDSP 포드 구매 필요) - F0, F1, F2, F3, F4, F7, G0, G4, H7, L0, L1, L4, L5, WB, WL 시리즈 지원
ver 9.5 12/2020	TI C28x	- 이전 Legacy 비트필드를 설치패키지에서 제외 - easyDSP 포드의 /BOOT 핀 운전 시점 변경 - 원활한 F2837xD/F2838xD CPU2 램부팅 사용을 위한 소스 파일 (easy28x_BitField_v9.5.c/ easy28x_DriverLib_v9.5.c)변경 **주의 사항** - F2837xD/F2838xD CPU2 램부팅 사용시 easy28x_BitField_v9.5.c/ easy28x_DriverLib_v9.5.c 사용 필수 **수정된 버그 :** - 메모리창 0 번지에 잘못된 변수 심볼이 표기됨 (v9.3, v9.4 만의 버그) - 2807x, 2837xS, 2837xD CPU1 : 부트롬을 위한 램 영역 오류 (v9.4 만의 버그)
ver 9.4 10/2020	TI C28x	- 차트 윈도우 관련 개선 (특히 대용량 데이타 표시에 용이) : 화면 갱신 주기 개선, 빠른 속도 ('Enable fast reading' 옵션 활성화 필요), 반복적 통신 실패시 Pause - 램 부팅 verify 속도 개선. 'Enable fast verifying' 옵션 활성화 필요. - 28002x, 2838x 플래시롬 대화상자에서 flashAPI wrapper 의 오토 보딩 효율 개선 - 플래쉬롬 대화상자에서 flashAPI wrapper 의 부팅 verify 생략으로 속도 개선 (28002x, 2837x, 2838x 에서는 이전 버전부터 생략하고 있었으며 나머지 MCU 에 대해서 이번에 적용) - 비트필드용 소스 파일(easy28x _bitfield_v9.4.c, easy28x_gen2_bitfield_v9.4.c)에서

		4 바이트/8 바이트 변수를 한번에 읽는 것으로 변경 -메뉴 표기 및 단축키 변경 (Serial Booting, ALT+S -> RAM Booting, ALT+R) **주의 사항 :** - 2838x CM : "easy2838x_cm_driverlib_v9.4.c" 사용 필수 !!!! **수정된 버그 :** - 차트 윈도우 : 특정 경우 out 파일 갱신시 차트 윈도우 오류 발생 - 트리 윈도우 : 변수 리스트에 의미없는 역참조 연산자 변수 등록 (v9.3 만의 버그) - 2838x : CPU1+CM 사용시 CPU1 측 프로젝트에서 프로그램 갱신시 CM 측 프로젝트에 갱신된 사항이 반영 안됨. - 2807x, 2837xS, 2837xD CPU1, 2837xD CPU2, 2838x CPU2 : 부트롬을 위한 램 영역 오류 - 2838x CM : 램 부팅 verify 때 오류 발생시, 오류 발생 번지를 잘 못 표기 - 2838x CM : section 의 시작번지가 64bit aligned 되어 있을 때, 플래쉬롬 라이팅 오류 발생
ver 9.3 6/2020	TI C28x	- F280x, F281x, F28044 에 대해서 새로운 BitField 소스 지원 - F2838xS, F2838D 지원하는 BitField 소스 지원 - 램 부팅시 사용자 코드 영역이 부트롬 및 easyDSP 사용 영역과 중첩되는 지 검토 및 경고 - 포인터 변수의 *(역참조) 연산자 지원 **수정된 버그 :** - 플래쉬롬 대화상자에서 특정 조건시 flashAPI wrapper 로 부팅하지 않음. - 와치창에서 갱신 주기 오류 - 2838x CM 의 플래시롬 대화상자에서 섹션 얼라이먼트 체크 오류
ver 9.2 4/2020	TI C28x	- TMS320F2838xS, TMS320F2838xD 지원 (driverlib 로만) - TMS320F28002x 지원 - driverlib/bitfield 에서 Rx 입력핀을 pullup 으로 설정하여 노이즈 내성 강화 - F2802x, F2802x0, F2803x, F2805x, F2806x 에 대해서 새로운 BitField 소스 지원 **수정된 버그 :** - 특정 경우 bin 파일 생성이 안되는 오류 - 'Reload *.out' 메뉴 실행 후 새로 로딩된 아웃파일에 맞춰 윈도우가 갱신되지 않음 - 특정 경우 Dwarf version 4 를 지원 못함 - 트리 윈도우에서 구조체/공용체의 포인터 변수를 구조체/공용체 변수로 인식

		- 28004x 플래시 대화상자에서 섹터 선택 오류 및 외부 클럭 없이는 동작하지 않음
ver 9.1 3/2020	TI C28x	- driverlib 지원 (28004x, 2807x, 2837xS, 2837xD) 및 main.c 예제 파일 제공 - 새로운 bitfield 기반 easyDSP 소스파일 및 main.c 예제 파일 제공 (28004x, 2807x, 2837xS, 2837xD, 2823x, 2833x, 2834x 에 대해서) - out 파일 재로딩 메뉴 제공 (SCI 부팅, flashrom 부팅 기능을 사용하지 않고 easyDSP 의 통신 기능만을 사용할 경우, 또는 easyDSP 를 디버거랑 같이 사용하는 경우, 사용자가 수동으로 갱신된 out 파일을 로딩할 수 있도록) - ELF-based Embedded Application Binary Interface (EABI) 지원 - 2837xD, 2837xS, 2807x, 28004x 플래쉬롬 대화상자에서 flashAPI wrapper 의 부팅 개선 **수정된 버그 :** - 트리 윈도우에 구조체 포인터 변수가 등록 - (v9.03 버전 버그) 2837xS, 2807x, 28004x 플래쉬롬 대화상자에서 flashAPI wrapper 의 부팅 오류
ver 9.03 1/2020	TI C28x	**수정된 버그 (ver 9 이후 버전만의 버그) :** - 프로젝트 폴더와 컴파일러 출력파일 (*.out) 폴더가 다른 경우 오동작 - 2837xS, 2837xD, 2807x, 28004x 경우 오토보딩 실패
ver 9.02 12/2019	TI C28x	- 프로젝트 생성시에 사용자가 debugging model (coff 또는 dwarf)을 설정하도록 함. 기존 프로젝트를 열 때에는 디폴트로 coff 가 설정됨.
ver 9.01 12/2019	TI C28x	**수정된 버그 :** - 일부의 경우, --symdebug:dwarf 로 컴파일된 out 파일을 --symdebug:coff 형식으로 해석
ver 9 12/2019	TI C28x	- 상위 컴파일러 지원 (버전 16 이상) - 컴파일 옵션 --symdebug:dwarf 사용 가능 - 향후 --symdebug:coff 에 대해서 easyDSP 성능 개선 및 버그 수정 (특히 변수 인식 관련) 지원이 없을 예정 !!!! 이후 버전의 업데이트 및 버그 수정 사항이 --symdebug:coff 에 대해서는 해당되지 않을 수 있습니다. - --symdebug:dwarf 사용시 typedef 로 정의된 변수의 타입은 typedef 이름으로 표기 (ex. Uint32)
ver 1 to ver 9		- ver 9 이전의 업데이트 이력은 easydsp@gmail.com 으로 문의 바랍니다.

ver. 1.0 8/1999		- 첫 릴리즈

5. 제한 사항

하기에 easyDSP 제한 사항을 명시합니다.

일반

1. Little endian 만 지원합니다.
2. easyDSP 는 SCI 또는 UART 의 통신 인터럽트로 MCU 와 데이타를 교환합니다. 멀티 코어의 경우 SEV, IPC 등을 사용하는 경우도 있습니다.
   따라서 MCU 의 리소스가 제한되어 통신 인터럽트에 시간이 적절히 할당되지 못할 경우, 제대로 동작하지 않을 수 있습니다.
3. C28x 에서만 기본형 변수에 대한 포인터 변수의 역 참조 연산자(*)가 지원됩니다.즉, 포인터를 가르키는 포인터 변수, 배열의 포인터 변수 등에는 지원되지 않습니다.
   다른 MCU 에서는 역참조 연산자가 지원되지 않습니다.
4. 화살표 연산자(->)는 지원되지 않습니다.
5. 구조체의 일종인 bit field 형식의 변수에 대하여 '쓰기'는 지원되지 않습니다. 읽기는 지원됩니다.
6. 다차원 배열의 경우 10 차원까지만 지원합니다.

기능 제한

MCU 별 easyDSP 의 기능 차이에 대해서 하기 표 참조 바랍니다. (지원 = o, 미지원 = x)
보호 기능 또는 보안 기능이 적용된 경우 플래시 프로그래밍이 불가능할 수 있습니다.
램부팅, 플래시 프로그래밍의 기능 제한 상세 사항은 각 MCU 별 해당 메뉴를 참조하세요.

벤더	MCU	모니터링	램부팅	플래시 프로그래밍	기타 제한 사항
TI	C28x	o	o	o	1. OTP 미지원
	AM263x	o	o (1)	o (1)	1. TI 제공 및 사용자 SBL 기반이므로 SBL 기능에 따라 제약됨.
	TM4C	o	x	o	1. EEPROM 미지원
	MSPM0	o	x	o (1)	1. MAIN 플래시 영역만 지원

ST	STM32	o	o (4)	o (1,2)	1. OTP memory, Data memory, Option bytes 억세스 지원 안함 2. Trust Zone, Secure MPU 미지원 3. MCU 내장 부트로더의 제한점 및 버그로 인한 easyDSP 제한점 4. Dual core MCU 경우 램부팅 미지원
Infienon	PSoC4	o	x	o (2)	1. easyDSP 는 UART 를 사용하여 MCU 와 통신하는 바, UART 가 지원되지 않는 PSoC4000 는 지원하지 않습니다 2. Single-application bootloader 구성에서만 플래시 프로그래밍 가능
	XMC1	o	x	x	
	XMC4	o	x	o	
Renesas	RA	o	x	o (1), x(2)	1. DLM 기능있는 MCU 경우 DLM 상태 변환 미지원 2. RA0 시리즈 MCU 는 플래시프로그래밍 지원 안함
	RX	o	x	o (1,2)	1. 보호기능이 적용된 (Area protection 또는 TM memory) 영역은 플래시 프로그래밍 미지원 2. RX64M, RX660, RX66T, RX71M, RX72T 시리즈는 option setting memory 영역 미지원
Toshiba	TX	o	x	o	
	TXZ3	o	x	o	
NXP	S32	o	x	o	EEPROM, UTEST 미지원
	LPC1x00	o	x	o	EEPROM 미지원

6. Pod 구성

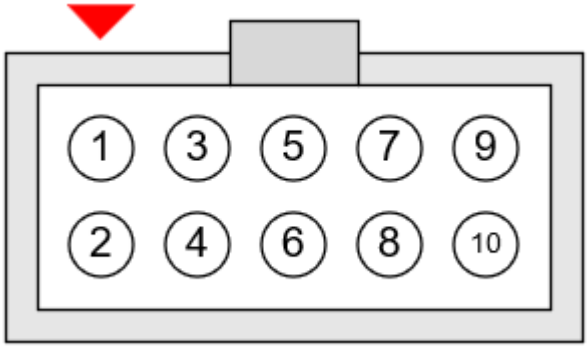
핀 구성

easyDSP Pod 의 신호선의 구성은 다음과 같습니다.

핀 간격은 2.54mm 입니다.

사용자 보드에서 easyDSP 용 커넥터로 BHS-01-10P 또는 XG4C-1031 을 사용하세요.

커넥터의 화살표에 유의하세요.



Pod 종류 1 또는 2 : TI C28x MCU 용

번호	이름	설명
1	RX	출력핀으로 MCU 의 RX 핀에 연결.
2	GND	전원 그라운드. easyDSP 포드내에서 10 번핀과 연결되어 있음.
3	TX	입력핀으로 MCU 의 TX 핀에 연결
4	VDD	전원핀으로 MCU 의 VDDIO 전위 연결 (ex : 3.3V)
5	/BOOT	pseudo open collector 타입 출력. MCU 부트 모드 진입시에만 Low 출력하고, 이외에는 신호 출력 없음.
6	reserved	연결하지 마세요
7	reserved	연결하지 마세요
8	reserved	연결하지 마세요
9	/RESET	pseudo open collector 타입 출력. MUC 리셋시에만 LOW 출력하고, 이외에는 신호 출력 없음.
10	GND	전원 그라운드. easyDSP 포드내에서 2 번핀과 연결되어 있음.

Pod 종류 3 : Arm 계열용 및 기타 코어 (RX)

번호	이름	설명
1	RX	출력핀으로 MCU 의 RX 핀에 연결
2	GND	전원 그라운드. easyDSP 포드내에서 10 번핀과 연결되어 있음.
3	TX	입력핀으로 MCU 의 TX 핀에 연결

4	VDD	전원핀으로 MCU 의 VDDIO 전위 연결 (ex : 3.3V, 1.8V)
5	/BOOT	pseudo open collector 타입 출력. MCU 부트 모드 진입시에만 Low 출력하고, 이외에는 신호 출력 없음
6	reserved	연결하지 마세요
7	BOOT	pseudo open emitter 타입 출력. MCU 부트 모드 진입시에만 High 출력하고, 이외에는 신호 출력 없음.
8	reserved	연결하지 마세요
9	/RESET	pseudo open collector 타입 출력. MUC 리셋시에만 LOW 출력하고, 이외에는 신호 출력 없음.
10	GND	전원 그라운드. easyDSP 포드내에서 2 번핀과 연결되어 있음.

/BOOT, BOOT 핀 설명

본 핀은 /RESET 핀과 연동되어 동작하며, MCU 리셋시 MCU 가 플래시로 부팅할 지, 아니면 부트모드로 진입할 지를 결정합니다.

MCU 가 플래시로 부팅할 때에는 (사용자 코드 수행), /BOOT, BOOT 핀은 신호를 출력하지 않으며,

MCU 가 easyDSP 로 인해 부트모드로 진입할 때에만 (램 부팅 또는 플래시 프로그래밍을 위해), /BOOT, BOOT 핀은 하기와 같이 신호를 출력합니다.

MCU	사용 부트핀	부트핀 동작
C28x XMC4 TX TXZ3 LPC1x00 S32	/BOOT	MCU 리셋시 /BOOT 핀은 Low 출력. MCU 리셋이 풀리고 약 1 초 뒤에 /BOOT 핀 신호 출력 없음.
STM32	BOOT	MCU 리셋시 BOOT 핀은 High 출력하며, 부트모드 진입 기간 동안 지속 출력 유지. 부트모드에서 나올 때에 (램부팅/플래쉬롬 대화상자에서 나올 때), BOOT 핀 신호 출력 없어짐.
AM2x TM4C MSPM0	BOOT	MCU 리셋시 BOOT 핀은 High 출력. MCU 리셋이 풀리고 약 1 초 뒤에 BOOT 핀 신호 출력 없음.
RA, RX	/BOOT	MCU 리셋시 /BOOT 핀은 Low 출력. RA : 부트 모드 진입 이후 Command acceptance phase 에 진입할 때 /BOOT 핀 신호 출력

		없어짐.
XMC1 PSOC4	미사용	미사용

/BOOT, BOOT 핀에 연결된 MCU 의 해당 핀을 사용자 프로그램에서 사용해야 할 경우에는 (예를 들어 EMIF 신호)
MCU 리셋 이후 초기 시간에 해당 핀의 전위를 입력 받아 (리셋 디폴트로 핀은 입력 IO),
이 값이 Low 또는 High 로 변경되기를 기다린 후, 즉 MCU 가 부트 모드에서 나온 이후,
해당 핀을 원하는 목적으로 설정한 후 사용할 수 있습니다.

LED 설명

두가지 상태 표시 LED 가 있으며 그 의미는 하기와 같습니다. 두 LED 모두 easyDSP 동작 중에는 ON 되어 있어야 합니다 (깜빡이지는 않음).
'DSP' 또는 'MCU' LED : easyDSP 커넥터를 통해 전원이 공급 중임을 표시. 즉, easyDSP 커넥터 4 번에서 VDD 가 공급되고 있음. 이는 결국 MCU 보드의 전원이 활성화됨을 의미함.
'USB' LED : easyDSP PC 프로그램이 easyDSP pod 하드웨어와 연결됨을 표시. 사용자가 easyDSP 프로그램에서 프로젝트를 오픈할 경우 연결됨.
주의) 광케이블 Pod 의 경우, PC 측 pod 의 DSP LED 와 MCU 측 Pod 의 USB LED 는 동작하지 않음.
LED 색깔은 의미 없음.

연결 순서

MCU 동작중 easyDSP 와의 물리적 연결/분리는 의도치 않은 MCU 리셋을 초래할 수 있으므로 금지하여 주십시오.
어쩔 수 없는 경우라면, 연결시에는 PC 에 먼저 연결하고 이후 DSP 에 연결, 분리시에는 DSP 에서 먼저 분리하고 이후 PC 에서 분리하여, 의도치 않은 MCU 리셋 발생 가능성을 최소화하여 주시기 바랍니다.

연결 방식

PC 와 easyDSP pod 간 직접 연결을 권장드리며 (USB 확장 포트 사용하지 말고), 필요시 새로운 USB 케이블을 사용하시기 바랍니다.

정격 사양

항목	포드 종류 1 : TI C28x 표준 절연 포드	포드 종류 2 : TI C28x 광케이블 절연 포드	포드 종류 3 : Arm Cortex-M 및 RX 표준 절연 포드
VDD 핀 인가 전압 범위	min 3, typ 3.3, max 5 [V]	왼쪽과 동일	min 1.65V, max 5V [V]
VDD 핀 추천 전압	3.3V	3.3V	MCU VDDIO (ex, 3.3V 또는 1.8V)
기타 입력 핀 전압	-0.5 .... VDD+0.5 [V]	왼쪽과 동일	왼쪽과 동일

VDD 핀 전류	max 3mA	max 50mA	max 10mA
1,2 차간 최소 절연 전압	2.0kVrms@1min	-	2.0kVrms@1min
동작 가능 주변 온도	5 .. 55 [°C]	왼쪽과 동일	왼쪽과 동일
보관 온도	-20 .. 65 [°C]	왼쪽과 동일	왼쪽과 동일
상대 습도 (non-condensing)	max 90% rH	왼쪽과 동일	왼쪽과 동일
크기 (케이블 제외)	82 x 56 x 21 mm^3	왼쪽과 동일(포드 2 개임)	81 x 42.5 x 21 mm^3
무게 (케이블 제외)	140 g	330 g	62 g
USB 인터페이스	USB 2.0 Hi-Speed	왼쪽과 동일	왼쪽과 동일

7. MCU 별 사용법

7.1 C28x

7.1.1 C28x 프로그래밍

7.1.1.1 일반 사항

비트필드/DriverLib easyDSP 통신 파일 제공

easyDSP 설치 폴더안에 'Source/C28x' 폴더는 하기와 같이 2 가지로 나뉩니다.

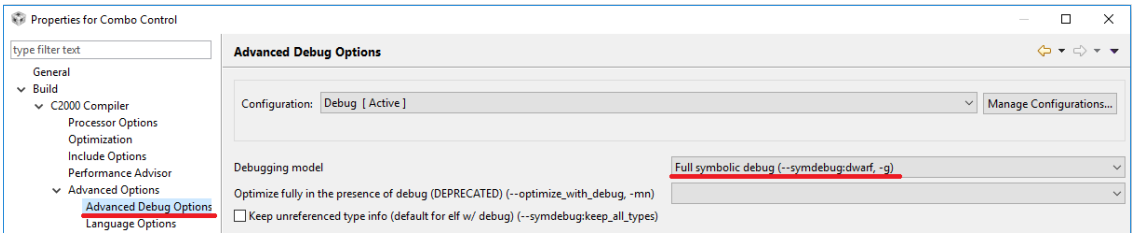
'BitField' 폴더는 비트필드 기반 소스이며, 'DriverLib' 폴더는 DriverLib(C28x Peripheral Driver Library) 기반 소스입니다.
사용자 프로젝트가 사용하는 함수 기반으로 easyDSP 도 동일하게 선택하여 주세요.
Bitfield 및 DriverLib 의 상세 사항은 TI 링크 를 참조하세요.
MCU 종류별 지원되는 방식은 하기 표와 같습니다.

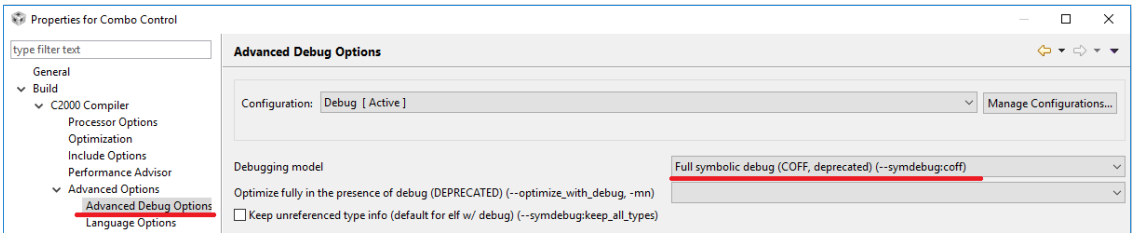
	Bitfield	DriverLib
F28001x F28002x	지원	지원

F28003x F28004x F2807x F2837x F2838x F28P55x F28P65x		
C2834x F2823x F2833x F281x F280x F28044 F282x0 F2802x F2803x F2805x F2806x	지원	미지원

디버깅 모델 옵션

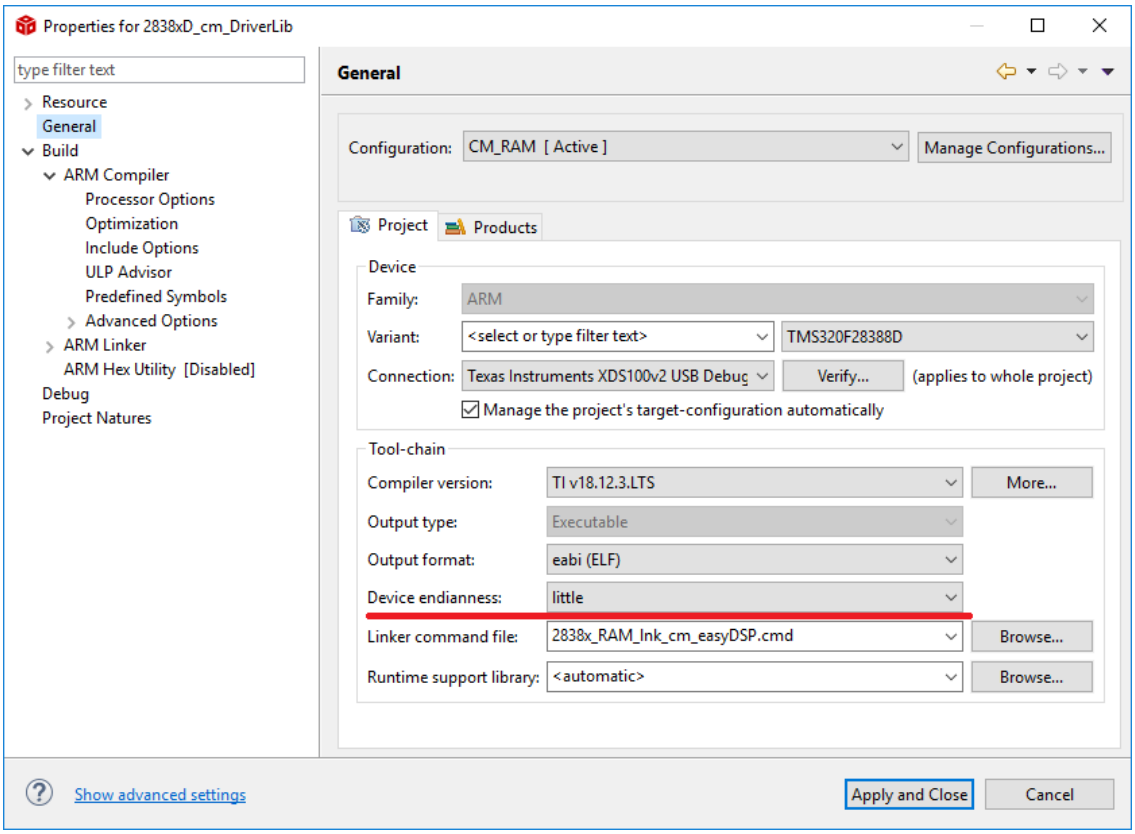
easyDSP 는 --symdebug :coff, --symdebug:dwarf 2 가지 컴파일 옵션을 지원합니다. 최신 컴파일러 (버전 16 이상)에서는 --symdebug:coff 옵션을 지원하지 않음에 유의하시기 바랍니다.
따라서 향후 --symdebug:coff 에 대한 지원이 제한될 예정이므로, 가급적--symdebug:dwarf 를 사용하시길 권장드립니다.
선택된 옵션을 그대로 easyDSP 의 project setting 에도 설정해야 함에 유의 부탁드립니다.





endian 옵션

little endian 만 지원합니다. 하기와 같이 설정하세요.



Generation-3 MCU 사용시 플래시롬 섹션 주소 관련 주의 사항

Gen3 MCU (ex : F2807x, F28001x, F28002x, F28003x, F28004x, F2837x, F2838x, F28Px)의 경우 플래시롬 사용시 플래시롬에 데이타가 저장될 섹션의 시작 주소가 최소 4 워드 또는 권장 8 워드에 얼라인되도록 요구하고 있습니다. 즉, C28x 코어의 경우 섹션 시작 주소의 최하위 번지가 0x0, 0x4, 0x8, 0xC 로 마감되어야 하며, Arm Cortex-M4 (ex, F2838x CM)의 경우, 0x0, 0x8 로 마감되어야 합니다. 아래 TI 에서 제공하는 링커 커맨드 파일을 보시면, 기본 섹션에 대해 ALIGN(*)을 적용하여 이미 반영되어 있음을 알 수 있습니다. 사용자가 섹션을 새로 생성할 경우에도 ALIGN(*)이 적용될 수 있도록 주의 부탁 드립니다 .

<TMS320F28388 CPU1/CPU2 경우>

```
SECTIONS
{
   codestart           : > BEGIN, ALIGN(8)
   .text               : >> FLASH1 | FLASH2 | FLASH3 | FLASH4, ALIGN(8)
   .cinit              : > FLASH4, ALIGN(8)
   .switch             : > FLASH1, ALIGN(8)
   .reset              : > RESET, TYPE = DSECT /* not used, */
   .stack              : > RAMM1

#if defined(__TI_EABI__)
   .init_array      : > FLASH1, ALIGN(8)
   .bss             : > RAMLS5
   .bss:output      : > RAMLS3
   .bss:cio         : > RAMLS5
   .data            : > RAMLS5
   .sysmem          : > RAMLS5
   /* Initalized sections go in Flash */
   .const           : > FLASH5, ALIGN(8)
#else
   .pinit           : > FLASH1, ALIGN(8)
   .ebss            : > RAMLS5
   .esysmem         : > RAMLS5
   .cio             : > RAMLS5
   /* Initalized sections go in Flash */
   .econst          : >> FLASH4 | FLASH5, ALIGN(8)
#endif
```

링커 옵션

TI 에서 제공하는 DSP28x_CodeStartBranch.asm 의 'code_start' section 을 사용하여, 부팅후 c_int00 을 수행하기 전에 와치독을 비활성화시키기를 권장 드립니다. 특히 초기화할 변수의 수가 매우 많은 경우 (특히 TMS320C28346 에서 큰 프로그램 사이즈에 의해 사용 변수가 많아질 질 경우), 와치독이 활성화된 상태에서 c_int00 을 수행하게 되면, c_int00 내에서 cinit 섹션의 변수를 초기화시키다가 많은 시간이 소요되게 되어 와치독에 의한 리셋이 발생되게 되어 부팅이 실패하게 됩니다. 링크시-ecode_start 옵션을 사용하세요.

7.1.1.2 멀티코어 주의 사항

멀티코어 MCU

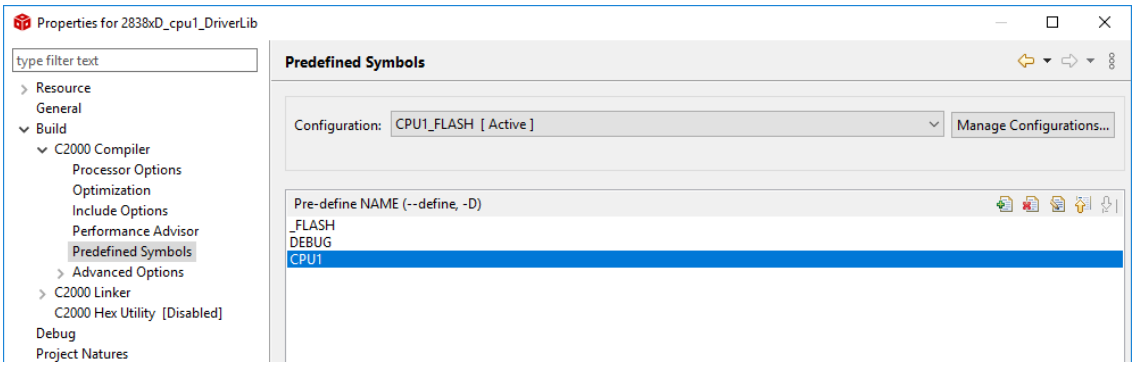
하기 제품이 대상입니다.

F28P65xD, F2827xD, F2838xS, F2838xD

컴파일러 Predefined Symbols 옵션

easyDSP 에서 제공하는 소스 파일은 CPU1, CPU2, CM, _FLASH 등 compiler 에서 선정의된 심볼을 참조하도록 작성되어 있습니다. 이러한 심볼은 멀티코어 MCU (ex, 2837x, 2838x 시리즈)를 사용하는데에 필수적인 조건입니다. F2837xD/F2838xD 의 CPU2 라면 CPU2 가 선정의되어야 하며, F2838x 의 CM 이라면 CM 이 선정의되어야 합니다.

보통 CCS 에서 자동 선정되지만, 혹시나 확인 부탁드립니다.



디버거 사용시

디버거 사용시에는 디버거가 각 코어에 메모리를 적절히 로딩하므로, easyDSP 의 멀티코어 부팅관련 함수 (easyDSP_Boot_Sync)를 동작시키지 말아야 합니다. easyDSP 가 제공하는소스파일 폴더의 main.c 의 서두 부분의 #define USE_DEBUGGER 을 참조하시기 바랍니다.

램부팅시 easyDSP 사용 리소스

멀티코어 MCU 의 램부팅을 위해 easyDSP 가 사용하는 리소스가 있습니다. 해당 리소스는 CPU2/CM 램 부팅 이전에 (easyDSP 제공 부팅 함수 easyDSP_Boot_Sync()호출 이전) 사용자 프로그램에서 사용되지 않도록 주의하세요. CPU2/CM 부팅 이후에는 사용하셔도 됩니다.

MCU	F2837xD	F2838xS F2838xD	F28P65xD
easyDSP 가 CPU2, CM 부팅시 사용하는 리소스	IPC_FLAG0 IPC_FLAG5 IPC_FLAG31	IPC_FLAG0 IPC_FLAG5 IPC_FLAG6 IPC_FLAG30 IPC_FLAG31 CPU1 to CPU2 MSGRAM1 CPU1 to CM MSGRAM1	IPC_FLAG0 IPC_FLAG5 CPU1 to CPU2 MSGRAM0

F2838x, F28P65xD CPU2, CM 플래시 부팅 시작 위치

easyDSP 제공 소스파일에서 하기와 같이 설정되어 있습니다.
F2838x 에서는 CPU2, CM 모두 플래시 sector 0 에서 부팅합니다.
F28P65xD CPU2 는 플래시 뱅크 3 에서 부팅합니다.
만약 다른 플래시 위치에서 부팅을 하고자 할 경우, easyDSP 제공 소스파일에서 easyDSP_Boot_Sync() 함수 내의 아래 부분 소스를 변경하여 주시기 바랍니다 .

F2838x BitField 사용시 :
ezDSP_Device_bootCPU2(BOOTMODE_BOOT_TO_FLASH_SECTOR0);
ezDSP_Device_bootCM(BOOTMODE_BOOT_TO_FLASH_SECTOR0);

F2838x DriberLib 사용시 :
Device_bootCPU2(BOOTMODE_BOOT_TO_FLASH_SECTOR0);
Device_bootCM(BOOTMODE_BOOT_TO_FLASH_SECTOR0);

F28P65xD BitField 사용시 : ezDSP_Device_bootCPU2(BOOTMODE_BOOT_TO_FLASH_BANK3_SECTOR0);
F28P65xD DriverLib 사용시 : Device_bootCPU2(BOOTMODE_BOOT_TO_FLASH_BANK3_SECTOR0);

F2838x, F28P65xD CPU2, CM 램부팅시 메모리 사용 제약

해당 MCU 는 SCI 를 통한 램부팅은 CPU1 에만 가능하며, CPU2, CM 에 대해서는 불가합니다.
easyDSP 에서 CPU2, CM 에 대해서도 SCI 를 통한 램부팅을 지원하기 위해, 먼저 CPU1 사용자 프로그램을 SCI 를 통해 램 부팅한 후, 별도의 Agent 프로그램으로 CPU2/CM 을 "IPC Message Copy to RAM" 모드로 부팅한 후, Agent 프로그램이 SCI 를 통해서 사용자 프로그램을 CPU2/CM 에 다운로딩하는 방식을 사용합니다.
따라서 Agent 프로그램이 동작할 일정 RAM 영역을 사용자 프로그램에서는 un initialized section 으로만 사용할 수 있습니다.
이러한 제약을 cmd 파일을 적절히 반영하시기 바랍니다.

	F2838x 램부팅 사용시 사용자 프로그램 메모리 제약	F28P65xD 램부팅 사용시 사용자 프로그램 메모리 제약
CPU1 사용자 프로그램	없음	없음
CPU2 사용자 프로그램	M1 RAM 영역 일부 (0x400 - 0x7F7)을 initialized section 으로 사용 불가	M1 RAM 영역 일부 (0x400 - 0x5FF)을 initialized section 으로 사용 불가
CM 사용자 프로그램	S0 RAM 영역 일부 (0x2000.0800 - 0x2000.0FFF)을 initialized section 으로 사용 불가	

F2837xD, F2838xD CPU2 램부팅 방식 변경 (easyDSP 소스파일 버전 11 부터) NEW

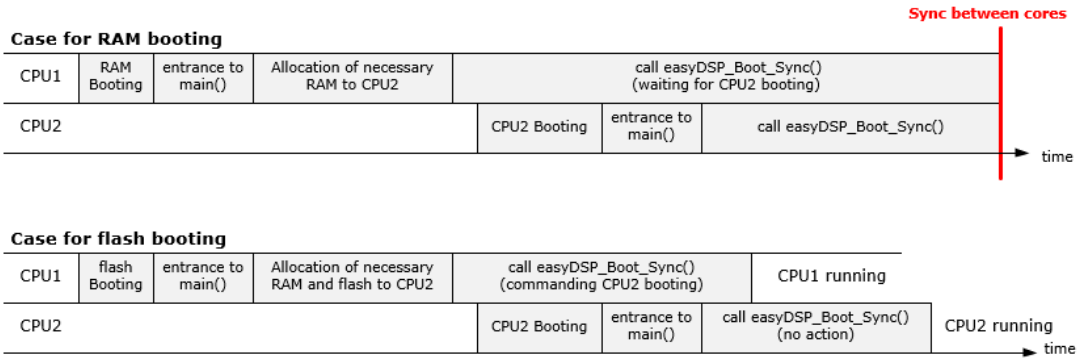
easyDSP 소스파일 버전 11 이전에는 F2837xD, F2838xD CPU2 램부팅 동작시
CPU1 용 easyDSP 소스파일내 easyDSP_SCIBootCPU2(void) 함수에서 모든 GSRAM (Global Shared RAM) 영역을 CPU2 에 할당한 뒤 CPU2 램부팅을 수행하고, 이후 다시 모든 GSRAM 영역을 CPU1 에 할당합니다.
따라서 CPU1 프로그램의 램부팅 관련 코드는 (easyDSP_SCIBootCPU2()함수의 .text 섹션) CPU1 LSRAM(Local Shared RAM)에 위치시켜야 하며,
CPU2 램부팅 이후 필요시 CPU1 에서 GSRAM 을 CPU2 에 할당해야하기에, 여러가지 주의사항이 수반되어야 합니다.

해당 사항에 대해서는 버전 11 이전 easyDSP 의 도움말을 참조하시기 바랍니다.

이러한 방식은 더이상 권장되지 않으므로, 가급적 easyDSP 소스파일 버전 11 및 이후 버전을 사용하시기 바랍니다.
새로운 방식에서는 더이상 easyDSP_SCIBootCPU2(void) 함수에서 GSRAM 을 CPU1 또는 CPU2 에 할당하지 않습니다.
대신, CPU1 프로그램 main.c 에서 easyDSP_SCIBootCPU2(void) 호출 이전에 CPU2 에서 사용하는 공유 메모리를 CPU2 에 할당합니다.
이로서 이전 버전의 제약 사항 및 고려 사항이 더 이상 필요하지 않습니다.

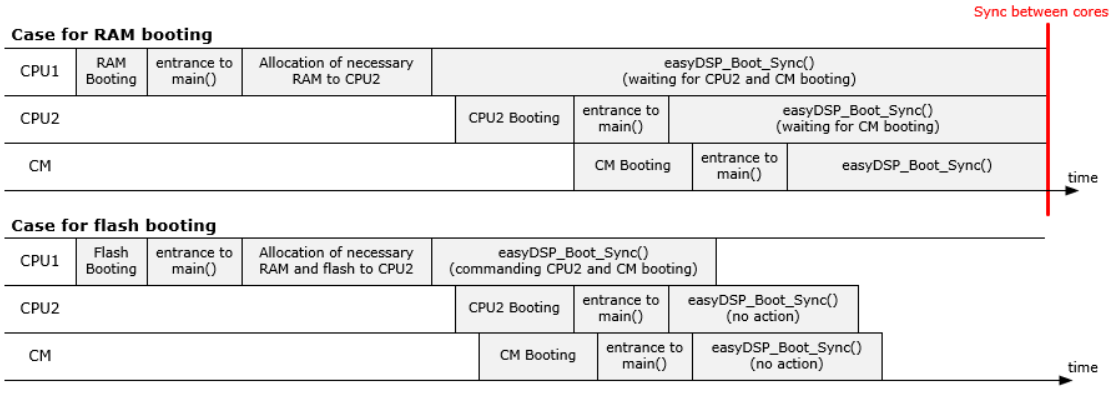
F2837xD, F28P65xD 의 CPU1, CPU2 부팅 순서 및 동기화

플래시 부팅의 경우 CPU1>CPU2 순으로 순차적 부팅을 하며 CPU 간 동기화는 수행되지 않습니다.
램부팅의 경우 CPU1>CPU2 순으로 부팅한 후 easyDSP_Boot_Sync() 함수 종료가 동기화됩니다.
CPU1 에서 CPU2 부팅을 시작하기 전 (즉, CPU1 에서 easyDSP_Boot_Sync 함수를 호출하기 전), CPU2 에 적절히 메모리를 할당해야함에 주의하세요.



F2838x 의 CPU1, CPU2, CM 의 부팅 순서 및 동기화

플래시 부팅의 경우 CPU1>CPU2>CM 순으로 순차적 부팅을 하며 CPU 간 동기화는 수행되지 않습니다.
램부팅의 경우 CPU1>CPU2>CM 순으로 부팅한 후 easyDSP_Boot_Sync() 함수 종료가 동기화됩니다.
CPU1 에서 CPU2/CM 부팅을 시작하기 전 (즉, CPU1 에서 easyDSP_Boot_Sync 함수를 호출하기 전), CPU2/CM 에 적절히 메모리를 할당해야함에 주의하세요.



F2838x 의 CPU2, CM 클럭 주파수는 ?

CPU1 이 CPU2 및 CM 을 부팅할 때, 각각 200Mhz, 125MHz 클럭 주파수를 세팅합니다. 이를 변경하려면 직접 해당 소스 부분을 수정하시기 바랍니다.

변경된 사용자 프로그램을 램부팅 또는 플래시에 프로그래밍할 경우

사용자가 도중에 프로그램을 변경하였다면 , 램부팅 내지는 플래시 프로그래밍 이후, 모든 코어에 대해서 변경된 프로그램의 out 파일을 사용하여 모니터링을 수행해야 합니다.
만약 각 코어에 연결된 easyDSP 가 동일한 PC 에 연결되어 있다면, 이러한 동작은 자동으로 진행됩니다.
예를 들어, CPU1 에 연결된 easyDSP 에서 CPU2 에 연결된 easyDSP 에 이를 반영하도록 자동으로 지령을 내립니다
만약 각 코어에 연결된 easyDSP 가 각기 다른 PC 에 연결되어 있다면, 이러한 작업은 MCU > Reload *.out 메뉴를 사용하여 사용자가 직접 수행해야 합니다.

7.1.1.3 Bitfield 사용시

비트필드 사용시

SCI 인터럽트 처리 루틴

easyDSP 는 MCU 의 SCI 통신을 사용하여 PC 와 MCU 간 통신을 구현합니다 . 따라서 사용자의 MCU 프로그램은 easyDSP 가 제공하는 SCI 인터럽트 처리 루틴 (Interrupt Service Routine,이하 ISR)을 포함하여야 합니다 .
관련된 파일은 아래와 같으며,easyDSP 가 설치된 폴더의 'source/C28x/BitField' 폴더에 위치합니다.
주의) F2838x CM 은 비트필드 기반 소스를 사용할 수 없으며, DriverLib 기반 소스를 사용하셔야 합니다 .

C28x 시리즈	ISR 프로그램 파일
F28001x F28002x F28003x F28004x F2807x F2837x F2838xS CPU1 F2838xD CPU1/CPU2 F28P55x F28P65x	easy28x_bitfield_v11.2.c easy28x_bitfield_v11.2.h
C2834x F2823x/2833x F2802x/2802x0 F2803x F2805x F2806x	easy28x_gen2_bitfield_v9.4.c easy28x_gen2_bitfield_v9.4.h

F280x F281x F28044	

ISR 프로그램 파일내의 대표적인 함수 이름 및 기능은 하기와 같습니다 .

easyDSP_SCI_Init(void) : SCI 통신 채널 초기화

easy_RXINT_ISR(void) : RX_INT 용 ISR

easy_TXINT_ISR(void) : TX_INT 용 ISR

easyDSP_SPI_Flashrom_Init(void) : C2834x 외부 SPI 플래쉬 부팅

easyDSP_Boot_Sync(void) : 2837xD/2838xS/2838xD 멀티코어의 부팅 및 동기화

사용자의 보드 환경에 맞춰서, 헤더 파일을 수정하셔야 함에 유의하세요 .

예를 들어 아래 선택은 F2807x 사용, CPU 클럭 150MHz, LSP 클럭 = CPU 클럭/4, easyDSP 통신 bps = 115200 으로 선택한 예제입니다.

```
#define F28P65xS                0
#define F28P65xD_CPU1           0
#define F28P65xD_CPU1_CPU2      0
#define F28002x                 0
#define F28003x                 0
#define F28004x                 0
#define F2807x                  0
#define F2837xS                 0
#define F2837xD_CPU1            0
#define F2837xD_CPU1_CPU2       0
#define F2838xS_CPU1            0
#define F2838xS_CPU1_CM         0
#define F2838xD_CPU1            0
#define F2838xD_CPU1_CPU2       0
#define F2838xD_CPU1_CM         0
#define F2838xD_CPU1_CPU2_CM    1
#define CPU_CLK                 150000000L
#define LSP_CLK                 (CPU_CLK/4)
#define BAUDRATE                115200L
```

주의 : MotorWare™ 에서는 클럭 분주를 하지 않게 되어 있습니다. 이 경우, LSP_CLK 을 CPU_CLK 과 동일하게 설정하셔야 합니다 .

참고로 easyDSP 가 통신에 사용하는 모든 변수의 이름은 ezDSP_*와 같은 식의 첨자가 추가되어 있습니다. easyDSP 사용중 ezDSP_ 로 시작되는 변수 값은 변경하지 마세요 .

인터럽트 네스팅 (Interrupt Nesting) 처리

C28x MCU 에서 하나의 ISR(인터럽트 서비스 루틴)이 수행될 때, 기본적으로는 다른 ISR 이 수행될 수 없게 되어 있습니다. 즉, 아무런 조치 없이는, easyDSP 용 ISR 이 수행될 때, 다른 중요한 사용자의 인터럽트가 수행될 수 없음을 의미합니다. 대부분의 경우 사용자는 리얼타임 제어를 유지하기 위해서, easyDSP ISR 수행여부에 상관없이, easyDSP 보다 상위 개념의 인터럽트를 수행시켜야 합니다 .

easyDSP ISR 소스 코드에서는 INT_NESTING_START, INT_NESTING_END 를 기본 제공하며, easyDSP ISR 의 인터럽트 순위는 최하위로 설정됩니다.

인터럽트간 다양한 우선순위를 구현하기 위해서는 TI 에서 예제 파일 (SW Prioritized ISR)를 사용하십시요. 이에 대해서는 하기 링크를 참조하셔서 사용자가 직접 코딩하시기 바랍니다. 즉, 사용자가 직접, easyDSP 의 ISR 즉 easy_RXINT_ISR ()의 첫단/마지막단 해당 부분을 수정하셔야 합니다 .

http://processors.wiki.ti.com/index.php/Interrupt_Nesting_on_C28x

플래쉬롬 동작시 easyDSP ISR 의 빠른 동작을 위해

프로그램이 플래쉬롬에서 작동하는 경우, easyDSP 의 ISR 함수의 빠른 동작을 위해서는 해당 코드를 램에서 동작시키는 것이 좋습니다. 이를 위해 헤더 파일에서 #progma 를 제공합니다. ramfuncs 또는.TI.ramfunc 섹션을 위한 제반 코딩은 TI 자료를 참조하십시오 .

easyDSP 제공 h 파일 에 하기 선언

```
#if (F2823x || F2833x || C2834x)
#pragma CODE_SECTION(easy_RXINT_ISR, "ramfuncs");
#else
#pragma CODE_SECTION(easy_RXINT_ISR, ".TI.ramfunc");
#endif
```

주의사항 1) 최신 DSP (ex, 2837x, 2807x, 28004x) 의 경우, 최신 TI support library version 에서는 ramfuncs 이 아니라 .TI.ramfunc 을 사용하고 있습니다. F28x_SysCtrl.c 파일 서두 부분을 보시면 어떤 것을 사용하는 지 파악 가능하니 이를 참조하여 주세요 .

주의사항 2) 특히 작성하신 프로그램이 "플래쉬에서 동작되는 것이면서 플래쉬를 프로그래밍"하는 것이라면 easyDSP 의 ISR 를 반드시 램에서 동작시켜야 합니다. TI 에서 제공되는 각종 flash API 함수가 수행되는 동안에는 flash 가 기본적으로 동작하지 않기 때문에 easyDSP ISR 코드가 flash 에서 수행되게 되면 오동작을 하는 것입니다 .

단일 코어 프로그래밍

MCU 프로그램 초기부분에 MCU 의 인터럽트(특히 시리얼 인터럽트 부분)를 easyDSP 와 통신할 수 있도록 설정해주어야 합니다. 아래에 사용 예제가 있습니다. 우선 사용자가 원하는 인터럽트를 설정합니다. 그 후에 easyDSP_SCI_Init()를 호출합니다 .

프로그램 설치 폴더의 source/C28x/BitField 폴더의 main_gen2.c 또는 main_gen3.c 파일을 참조하십시요 .
easyDSP_SCI_Init()에서는
1) SCI 관련 레지스터를 easyDSP 용도에 맞게 설정
2) SCI 인터럽트를 사용할 수 있도록 관련 레지스터를 설정합니다 .

```
#include " easy28x_bitfield_v11.2.h" 또는....
#include " easy28x_gen2_bitfield_v9.4.h"
main(void) {


    // 사용자의 인터럽트 관련 설정 이후, while(1) 이전 호출
    easyDSP_SCI_Init();

    while(1) {
    }
}
```

C2834x 프로그래밍 : 외부 flash rom 설정

2834x 의 경우에는 내부 플래쉬 롬이 없으므로, 외부 SPI 용 플래쉬 롬을 지원합니다. 현재 지원되는 플래쉬 롬은, ATMEL 사의 AT25DF021(2M bit), AT25DF041(4M bit), AT26DF081(8M bit), AT25DF321(32M bit) 및 Numonyx 사의 M25P20(2M bit), M25P40(4M bit), M25P80(8M bit), M25P16(16M bit), M25P32(32M bit)입니다. 이를 위한 DSP 의 SPI-A 포트 설정이 필요합니다 .
프로그램 설치 폴더의 source/C28x/BitField 폴더의 main_gen2.c 파일을 참조하십시요 .

```
#include " easy28x_gen2_bitfield_v9.4.h"
main(void) {


    // 사용자의 인터럽트 관련 설정 이후 호출
    easyDSP_SCI_Init();
    // 외부 플래쉬롬 사용을 위한 SPI-A 포트 설정
    easyDSP_SPI_Flashrom_Init();

    while(1) {
    }
```

```
}
```

F2837xD, F28P65xD, F2838xD 멀티코어 프로그래밍

싱글 코어 MCU 와 동일하게, CPU1/CPU2 모두에 대해서 헤더 파일을 설정하고 easyDSP_SCI_Init()함수를 호출합니다 .
다른 부분은 CPU2, CM 을 부팅시키고 CPU 간 동기를 맞추기 위한 easyDSP_Boot_Sync() 함수의 호출 부분입니다. easyDSP_Boot_Sync() 함수가 호출되면 CPU1 은 CPU2, CM 을 부팅시킵니다. 램 부팅시에는 CPU2, CM 이 램부팅되기까지 CPU1 은 기다립니다. 플래시 부팅시에는 CPU2, CM 에 플래시 부팅 지령을 내린 후 바로 CPU1 프로그램이 후속 진행됩니다 .
easyDSP_Boot_Sync() 함수는 모든 CPU 에서 호출되어야 하며, 기타 사항은 프로그램 설치 폴더의 source/C28x/BitField 폴더 안에 main_gen3.c 파일을 참조하십시요 .

```
#include "easy28x_bitfield_v11.2.h"
main(void) {

    InitSysCtrl();

    // CPU1 프로그램이라면, easyDSP_Boot_Sync() 호출 이전에
    // CPU2, CM 에 필요한 공유 메모리를 할당하는 코딩 삽입

    // easyDSP_SCI_Init() 이전 호출
    easyDSP_Boot_Sync();

    easyDSP_SCI_Init();


    while(1) {
    }
}
```

7.1.1.4 DriverLib 사용시

SCI 인터럽트 처리 루틴

easyDSP 는 MCU 의 SCI 통신을 사용하여 PC 와 MCU 간 통신을 구현합니다 .따라서 사용자의 MCU 프로그램은 easyDSP 가 제공하는 SCI 인터럽트 처리 루틴를 포함하여야 합니다 .
관련된 파일은 아래와 같으며, easyDSP 가 설치된 폴더의 'source/C28x/DriverLib' 폴더에 위치합니다.

C28x 시리즈	ISR 프로그램 파일
F28001x	easy28x_DriverLib_v11.2. c

F28002x F28003x F28004x F2807x F2837x F2838x CPU1 또는 CPU2 F28P55x F28P65x	easy28x_DriverLib_v11.2. h
F2838x CM	easy28x_cm_DriverLib_v10.1.c easy28x_cm_DriverLib_v10.1.h

ISR 프로그램 파일내의 대표적 함수 이름 및 기능은 하기와 같습니다 .
easyDSP_SCI_Init(void) : SCI 통신 채널 초기화
easyDSP_UART_Init (void) : UART 통신 채널 초기화 (F2838x 의 CM 대상)
easy_RXINT_ISR (void) : RX_INT 용 ISR
easyDSP_Boot_Sync(void) : F2837xD, F2838xS, F2838xD 멀티 코어 부팅 및 동기화

사용자의 보드 환경에 맞춰서, 헤더 파일을 수정하셔야 함에 유의하세요 .
예를 들어 아래 선택은 F2807x 사용, easyDSP 통신 bps = 115200 으로 선택한 예제입니다 .

```
#define F28002x                 0
#define F28003x                 0
#define F28004x                 0
#define F2807x                  1
#define F28P65xS                0
#define F28P65xD_CPU1           0
#define F28P65xD_CPU1_CPU2      0
#define F2837xS                 0
#define F2837xD_CPU1            0
#define F2837xD_CPU1_CPU2       0
#define F2838xD_CPU1            0
#define F2838xD_CPU1_CM         0
#define BAUDRATE                115200L
```

주의) device.h 파일안의 DEVICE_LSPCLK_FREQ 상수값이 사용 시스템에 매칭되어야 합니다. 이 상수는 easyDSP 통신의 SCI 보드레이트 설정에 사용됩니다.
참고로 easyDSP 가 통신에 사용하는 모든 변수의 이름은 ezDSP_*와 같은 식의 첨자가 추가되어 있습니다. easyDSP 사용중 ezDSP_로 시작되는 변수 값은 변경하지 마세요 .

인터럽트 네스팅 (Interrupt Nesting) 처리

C28x MCU 에서 하나의 ISR(인터럽트 서비스 루틴)이 수행될 때, 기본적으로는 다른 ISR 이 수행될 수 없게 되어 있습니다. 즉, 아무런 조치 없이는, easyDSP 용 ISR 이 수행될 때, 다른 중요한 사용자의 인터럽트가 수행될 수 없음을

의미합니다. 대부분의 경우 사용자는 리얼타임 제어를 유지하기 위해서, easyDSP ISR 수행여부에 상관없이, easyDSP 보다 상위 개념의 인터럽트를 수행시켜야 합니다 .
easyDSP ISR 소스 코드에서는 INT_NESTING_START, INT_NESTING_END 를 기본 제공하며, easyDSP ISR 의 인터럽트 순위가 최하위로 설정됩니다.
인터럽트간 다양한 우선순위를 구현하기 위해서는 TI 에서 예제 파일 (SW Prioritized ISR)를 사용하십시요. 이에 대해서는 하기 링크를 참조하셔서 사용자가 직접 코딩하시기 바랍니다. 즉, 사용자가 직접, easyDSP 의 ISR 즉 easy_RXINT_ISR ()의 첫단/마지막단 해당 부분을 수정하셔야 합니다 .
http://processors.wiki.ti.com/index.php/Interrupt_Nesting_on_C28x

플래쉬롬 동작시 ISR 의 빠르고 안정적인 동작을 위해

프로그램이 플래쉬롬에서 작동하는 경우, easyDSP 의 ISR 함수의 빠른 동작을 위해서는 해당 코드를 램에서 동작시키는 것이 좋습니다.이를 위해 헤더 파일에서 # progma 를 제공합니다. TI.ramfunc 섹션을 위한 제반 코딩은 TI 자료를 참조하십시오 .

easyDSP 제공 헤더파일 에 하기 선언
#pragma CODE_SECTION(easy_RXINT_ISR, ".TI.ramfunc");

주의사항)특히 작성하신 프로그램이 " 플래쉬에서 동작되는 것이면서 플래쉬를 프로그래밍 "하는 것이라면 easyDSP 의 ISR 를 반드시 램에서 동작시켜야 합니다. TI 에서 제공되는 각종 flash API 함수가 수행되는 동안에는 flash 가 기본적으로 동작하지 않기 때문에 easyDSP ISR 코드가 flash 에서 수행되게 되면 오동작을 하는 것입니다 .

단일 코어 프로그래밍

DSP 프로그램중 초기 부분에 DSP 의 인터럽트 (특히 시리얼 인터럽트 부분)를 easyDSP 와 통신할 수 있도록 설정해주어야 합니다 . 아래에 사용 예제가 있습니다 . 우선 사용자가 원하는 인터럽트를 설정합니다 .그 후에 easyDSP_SCI_Init()를 호출합니다 .
프로그램 설치 폴더의 source/C28x/DriverLib 폴더안에 main.c 파일을 참조하십시요 .
easyDSP_SCI_Init()에서는
1) SCI 관련 레지스터를 easyDSP 용도에 맞게 설정
2) SCI 인터럽트를 사용할 수 있도록 관련 레지스터를 설정합니다 .

```
#include "easy28x_DriverLib_v11.2 .h"
main(void) {

   // 사용자의 인터럽트 관련 설정 이후, while(1) 이전 호출
   easyDSP_SCI_Init();


   while(1) {
   }
}
```

멀티코어 프로그래밍 : F28P65xD, F2837xD, F2838xS, F2838xD 의 CPU1 또는 CPU2 프로그램

싱글 코어 MCU 와 동일하게, CPU1/CPU2 모두에 대해서 헤더 파일을 설정하고 easyDSP_SCI_Init() 함수를 호출합니다.
다른 부분은 CPU2 또는 CM 를 부팅시키고 각 코어간 동기화를 위한 easyDSP_Boot_Sync() 함수입니다.
easyDSP_Boot_Sync() 함수가 호출되면 CPU1 은 CPU2/CM 을 부팅시킵니다. 램부팅시에는 CPU2/CM 이 부팅되기까지 CPU1 은 기다립니다. 플래쉬롬 부팅시에는 CPU2/CM 에 부팅 지령을 내린 후 바로 CPU1 프로그램이 후속 진행됩니다 .
easyDSP_Boot_Sync() 함수는 CPU1/CPU2 모두에서 호출되어야 하며, 기타 사항은 프로그램 설치 폴더의 source/C28x/driverlib 폴더안에 main.c 파일을 참조하십시요 .

```
#include "easy28x_DriverLib_v11.2 .h"
main(void) {

    Device_init ();

    // CPU1 프로그램이라면, easyDSP_Boot_Sync() 호출 이전에
    // CPU2, CM 에 필요한 공유 메모리를 할당하는 코딩 삽입

    //  easyDSP_SCI_Init() 이전에 하기 함수 호출
    easyDSP_Boot_Sync();

    // 사용자의 인터럽트 관련 설정 이후, while(1) 이전 호출
    easyDSP_SCI_Init();

    while(1) {
    }
}
```

멀티코어 프로그래밍 : F2838x CM 프로그램

싱글 코어 MCU 와 유사하게, 헤더 파일을 설정하고 easyDSP_UART_Init () 함수를 호출합니다.
다른 부분은 CPU1/CM 간 동기화를 위한 easyDSP_Boot_Sync() 함수입니다.
기타 사항은 프로그램 설치 폴더의 source/C28x/driverlib 폴더안에 main_cm.c 파일을 참조하십시요 .

```
#include "easy28x_cm_DriverLib_v10.1.h "
main(void) {

    CM_init();

    // CM_init() 이후 , easyDSP_UART_Init()이전 호출
    easyDSP_Boot_Sync();

    easyDSP_UART_Init();

    while(1) {
    }
}
```

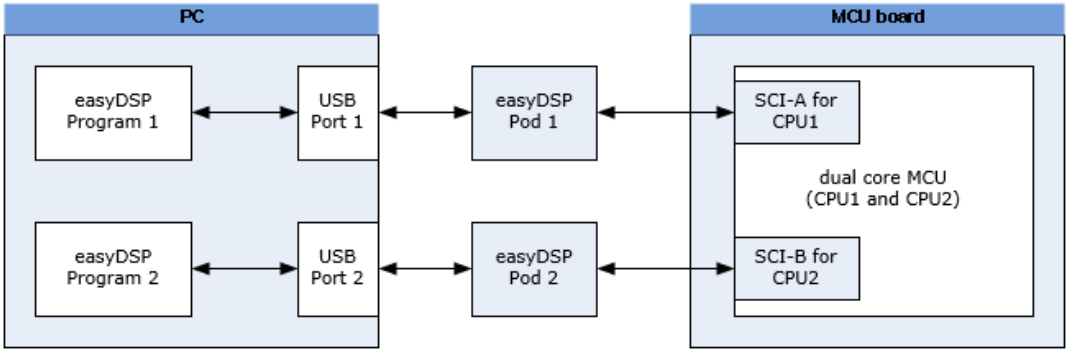
7.1.1.5 F2837xD, F28P65xD 사용

easyDSP 연결

Dual CPU (CPU1, CPU2)와 통신하기 위해서는 2 개의 easyDSP pod 가 필요하며 또한 2 개의 easyDSP 프로그램을 실행하여 각각 연결하여야 합니다. 하기 그림을 참조하십시오. easyDSP 프로그램은 다중 실행이 가능하며, 첫번째로 실행한 프로그램의 이름은 easyDSP, 두번째로 실행한 프로그램은 easyDSP(2)의 식으로 표시됩니다. 첫번째로 실행한 easyDSP 프로그램을 CPU1 에 연결, 두번째로 실행한 프로그램을 CPU2 에 연결하기 위해서는 특별한 주의가 필요합니다. 먼저 easyDSP pod 한 개를 PC 에 연결한 후, 이 easyDSP pod 를 CPU1 에 해당되는 SCI-A 포트에 연결합니다. 그리고 easyDSP 프로그램을 실행하여 사용자 프로젝트를 오픈하면, 이 상태에서 PC 에 연결된 easyDSP pod 는 한 개이므로, 당연히 해당 사용자 프로젝트는 CPU1 에 연결됩니다. 이 후 easyDSP pod 를 PC 에 추가로 연결하고 이를 CPU2 SCI-B 포트에 연결하고 easyDSP 프로그램을 실행 한 후, 사용자 프로젝트를 오픈하면, 2 번째 easyDSP 프로그램의 사용자 프로젝트는 CPU2 에 연결되게 됩니다.

주의) 한개의 easyDSP pod 를 사용할 경우에도, CPU1/CPU2 에 대해 램 부팅, 플래쉬롬 라이팅이 가능합니다. 단, CPU2 에 대한 모니터링을 할 수가 없습니다 .

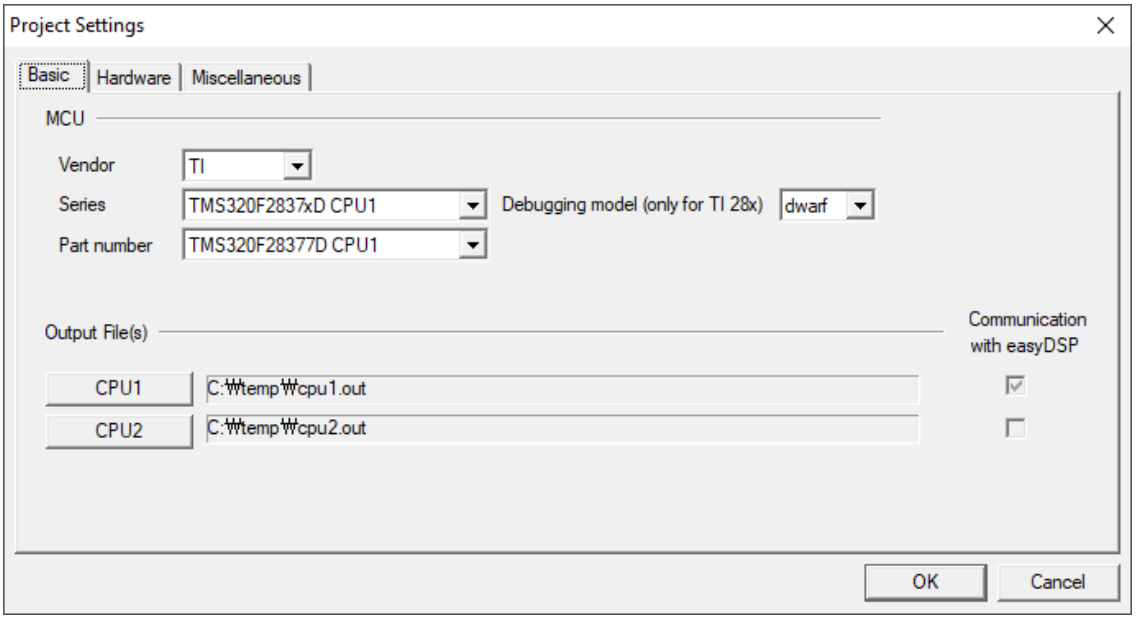
주의) CPU1, CPU2 프로젝트간 원활한 상호 작용을 위해 동일한 PC 에서 사용하시기를 권장 드립니다.



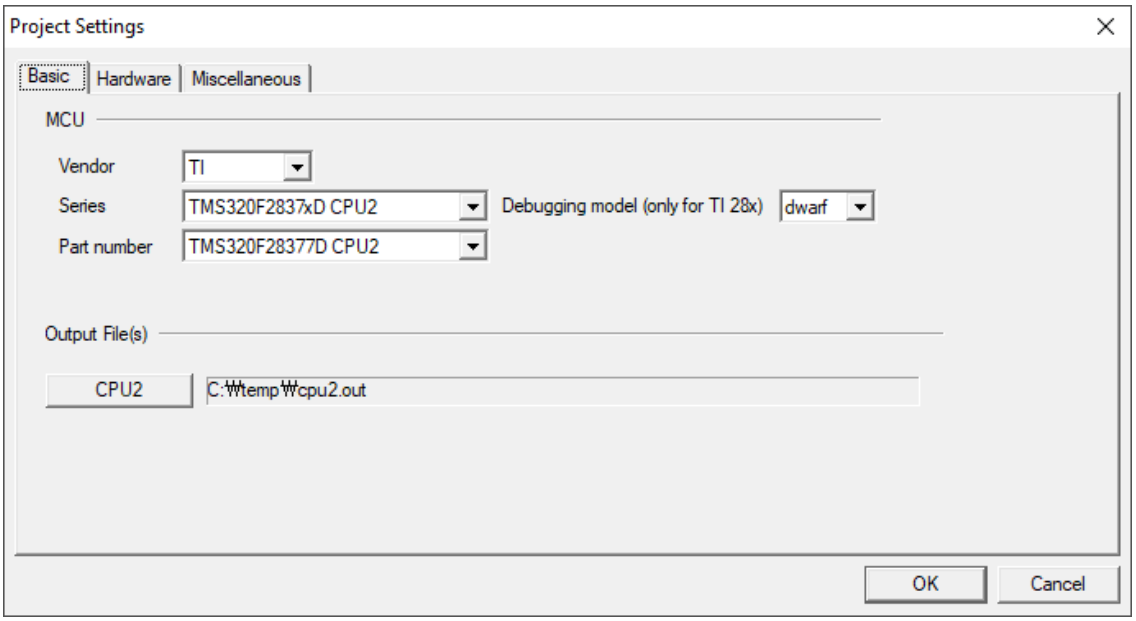
프로젝트 생성

두 개의 easyDSP 프로그램이 활성화되면 , 각각 easyDSP , easyDSP (2) 의 이름을 가지게 됩니다. CPU1 에 연결된 easyDSP 프로그램에서 프로젝트를 생성할 때 , CPU1 에서 사용하는 out 파일 및 CPU2 에서 사용하는 out 파일 모두 지정해야 합니다 . CPU2 의 부팅은 CPU1 에서 수행하기 때문에,즉 CPU2 의 부팅도 CPU1 에 연결된 easyDSP 프로그램에서 수행하기 때문에 그렇습니다.만약 CPU2 에 사용하는 out 파일을 지정하지 않는 경우에는 CPU2 를 사용(부팅)할 수 없습니다 . 또한 easyDSP 와의 통신은 CPU1 으로 고정됩니다.하기 그림을 참조하세요 . CPU2 에 연결된 easyDSP 프로그램에서 프로젝트를 생성할 때는 CPU2 에서 사용하는 out 파일만을 지정합니다. 이 out 파일은 CPU1 용 프로젝트에서 지정된 CPU2 용 out 파일과 동일해야 합니다.

<easyDSP 프로그램 1>



<easyDSP 프로그램 2>



램 부팅 및 플래시 프로그래밍 동작은 CPU1 에 연결된 easyDSP 에서

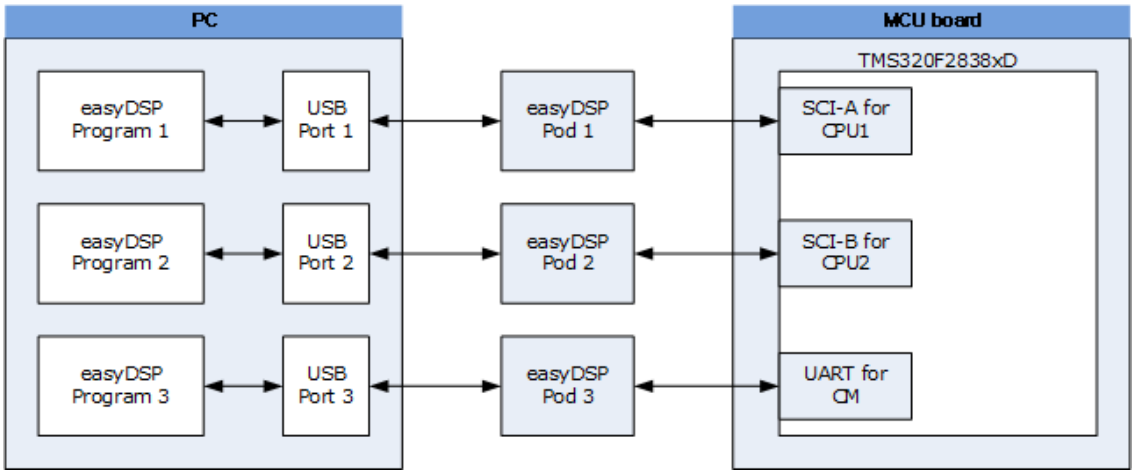
CPU1, CPU2 의 램 부팅 및 플래시 프로그래밍 , MCU 리셋 동작은 모두 CPU1 에서, 즉 CPU1 에 연결된 easyDSP 프로그램에서 수행됩니다. 단 부팅에 대한 Verify 동작은 각각의 프로그램에서 수행됩니다 .
CPU1 에서 램부팅, 플래시 프로그래밍 수행시 CPU2 에 연결된 easyDSP 의 통신은 잠시 멈추게 됩니다. (단, CPU1, CPU2 용 easyDSP 가 동일 PC 에 연결된 경우)

동작	easyDSP program 1	easyDSP program 2
CPU1, CPU2 램부팅	수행	미수행
CPU1, CPU2 램부팅에 대한 verify	CPU1 에 대해서만 수행	CPU2 에 대해서만 수행
CPU1, CPU2 플래시 제반 동작	수행	미수행
CPU 리셋	수행	미수행

7.1.1.6 F2838xD 사용

easyDSP 연결

multi core CPU TMS320F2838xD (CPU1, CPU2, CM)와 통신하기 위해서는 3 개의 easyDSP pod 가 필요하며 또한 3 개의 easyDSP 프로그램을 실행하여 각각 연결하여야 합니다. 하기 그림을 참조하십시오. easyDSP 프로그램은 다중 실행이 가능하며, 첫번째로 실행한 프로그램의 이름은 easyDSP, 두번째로 실행한 프로그램은 easyDSP(2), 세번째로 실행한 프로그램은 easyDSP(3)의 식으로 표시됩니다. 첫번째로 실행한 easyDSP 프로그램을 CPU1 에 연결, 두번째로 실행한 프로그램을 CPU2 에 연결, 세번째로 실행한 프로그램을 CM 에 연결하기 위해서는 특별한 주의가 필요합니다. 먼저 easyDSP pod 한 개를 PC 에 연결한 후, 이 easyDSP pod 를 CPU1 에 해당되는 SCI-A 포트에 연결합니다. 그리고 easyDSP 프로그램을 실행하여 사용자 프로젝트를 오픈하면, 이 상태에서 PC 에 연결된 easyDSP pod 는 한 개이므로, 당연히 해당 사용자 프로젝트는 CPU1 에 연결됩니다. 이 후 easyDSP pod 를 PC 에 추가로 연결하고 이를 CPU2 SCI-B 포트에 연결하고 easyDSP 프로그램을 실행 한 후, 사용자 프로젝트를 오픈하면, 2 번째 easyDSP 프로그램의 사용자 프로젝트는 CPU2 에 연결되게 됩니다. 마찬가지로 세번재 프로그램을 CM 에 연결합니다.
TMS320F2838xS 의 경우 CPU1/CM 의 2 가지 코어뿐이므로 2 개의 easyDSP pod 로 통신 가능합니다.
주의) 한개의 easyDSP pod 를 사용할 경우에도, CPU1/CPU2/CM 에 대해 램 부팅, 플래쉬롬 라이팅이 가능합니다. 단, CPU2/CM 에 대한 모니터링을 할 수가 없습니다 .
주의) CPU1, CPU2, CM 프로젝트간 원활한 상호 작용을 위해 동일한 PC 에서 사용하시기를 권장 드립니다.

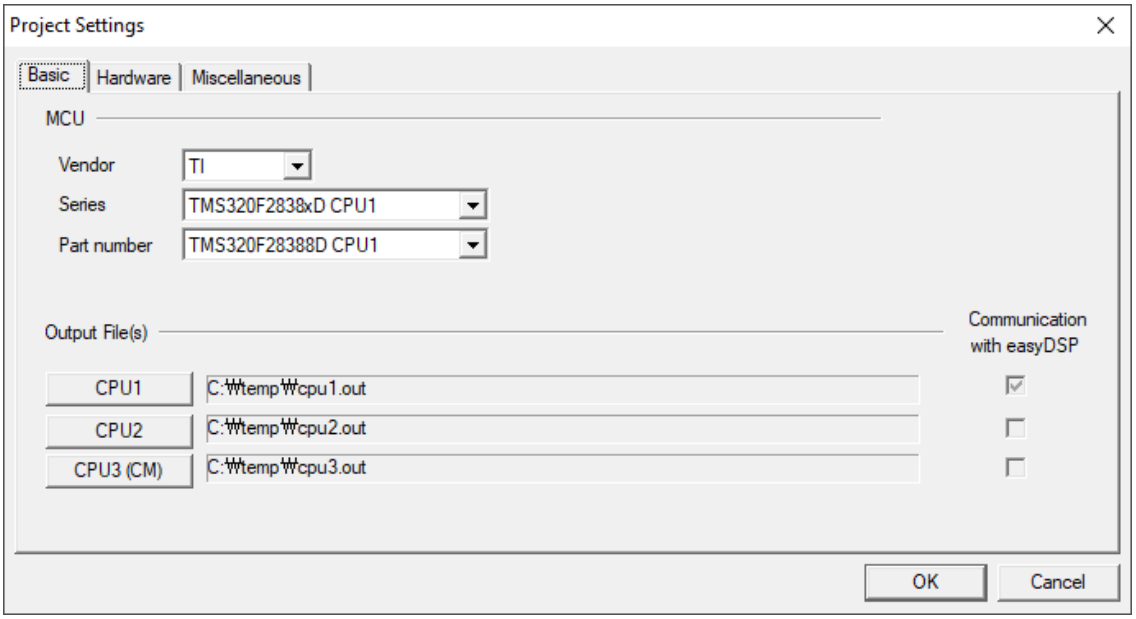


프로젝트 생성

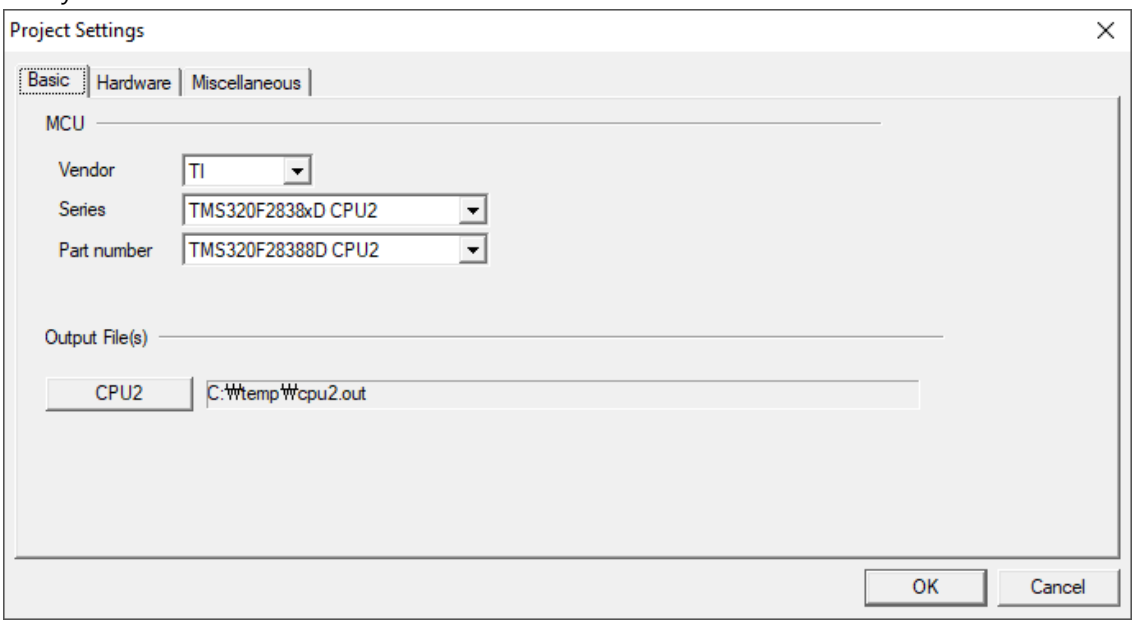
세 개의 easyDSP 프로그램이 실행되면 , 각각 easyDSP , easyDSP(2), easyDSP(3) 의 이름을 가지게 됩니다. CPU1 에 연결된 easyDSP 프로그램에서 프로젝트를 생성할 때 , CPU1 에서 사용하는 out 파일 및 CPU2/CM 에서 사용하는 out 파일들을 모두 지정해야 합니다 . CPU2/CM 의 램 부팅 및 플래시 프로그래밍은 CPU1 에서 수행하기 때문에,즉 CPU2/CM 의 부팅도 CPU1 에 연결된 easyDSP 프로그램에서 수행하기 때문에 그렇습니다.만약 CPU2/CM 를 사용하지 않는다면 out 파일을 지정하지 않아도 됩니다. 또한 easyDSP 와 통신은 CPU1 으로 고정됩니다. 하기 그림을 참조하세요 .

CPU2/CM 에 연결된 easyDSP 프로그램에서 프로젝트를 생성할 때는 CPU2/CM 각각 에서 사용하는 out 파일만을 지정합니다. 이 out 파일은 CPU1 용 프로젝트에서 지정된 CPU2/CM 용 out 파일과 동일해야 합니다.

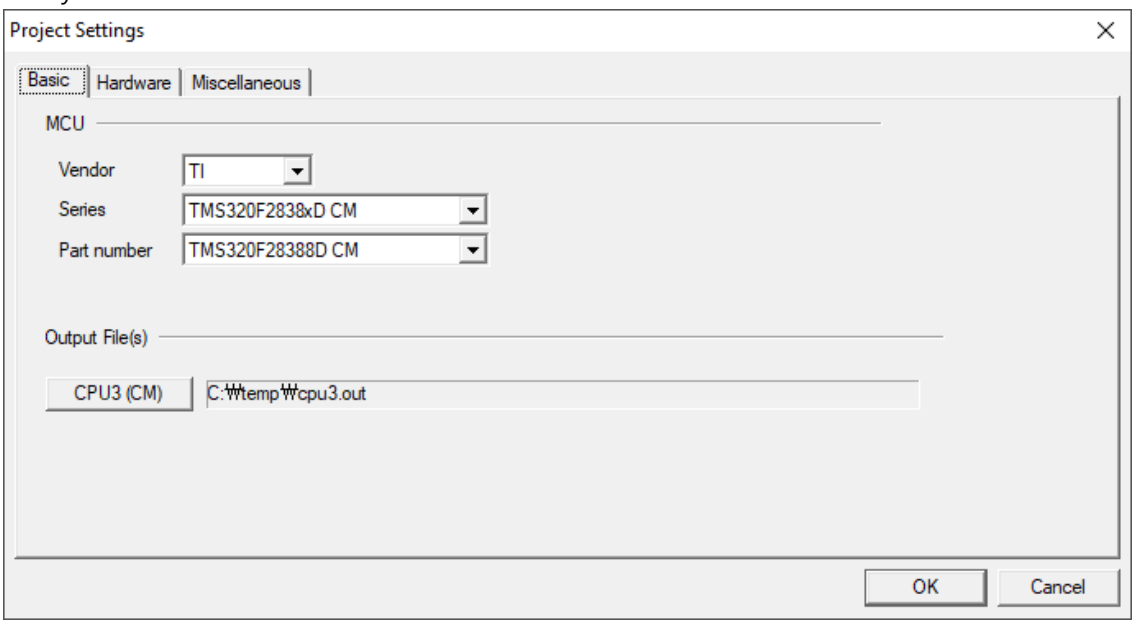
<easyDSP 프로그램 1>



<easyDSP 프로그램 2>



<easyDSP 프로그램 3>



램 부팅 및 플래쉬롬 라이팅 동작은 CPU1 에 연결된 easyDSP 에서

CPU1, CPU2, CM 의 램부팅 및 플래시 프로그래밍 동작은 모두 CPU1 에서, 즉 CPU1 에 연결된 easyDSP 프로그램에서 수행됩니다. 단지 부팅에 대한 Verify 동작만 각각의 프로그램에서 수행됩니다 .
CPU1 에서 램부팅, 플래시 프로그래밍 수행시 CPU2, CM 에 연결된 easyDSP 의 통신은 잠시 멈추게 됩니다. (단, CPU1, CPU2, CM 용 easyDSP 가 동일 PC 에 연결된 경우)

동작	easyDSP program 1	easyDSP program 2	easyDSP program 3
CPU1, CPU2, CM 램부팅	수행	미수행	미수행

CPU1, CPU2, CM 램부팅 베리파이	CPU1 에 대해서만 수행	CPU2 에 대해서만 수행	CM 에 대해서만 수행
CPU1, CPU2, CM 플래시 제반 동작	수행	미수행	미수행
CPU 리셋	수행	미수행	미수행

7.1.2 C28x 보드 설정

7.1.2.1 F28P65x

하기는 MCU 공장 출하 상태 설정값을 기반으로 설명드립니다.

만약 User OTP 의 BOOTPIN_CONFIG, BOOTDEF 값을 변경하여 사용자 부트 모드를 사용할 경우에는 적절히 하기 내용을 변경, 사용하시기 바랍니다.

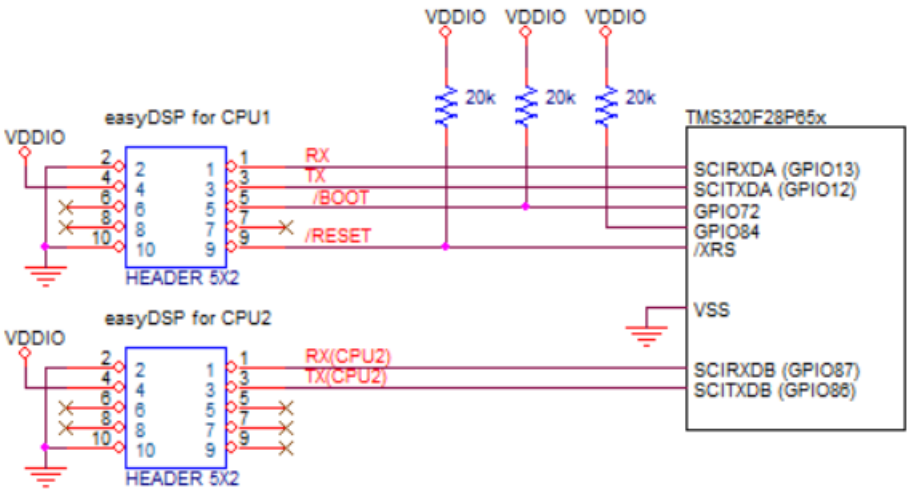
MCU 는 리셋시 2 개 핀의 전위 상태에 따라 다음 표와 같이 부트 동작 모드를 결정합니다.

부트모드	**GPIO72 (기본 부트모드 선택핀 1)**	**GPIO84 (기본 부트모드 선택핀 2)**
Parallel IO	0	0
SCI / Wait Boot (RAM 부트)	0	1
CAN	1	0
Flash / USB	1	1

easyDSP 는 SCI 부트모드(RAM 부트) 및 Flash 부트모드만을 사용하므로,

리셋시 GPIO84 는 항상 High 가 되도록 풀업 저항을 연결하며, GPIO72 은 easyDSP pod 의 /BOOT 신호에 연결되도록 합니다.

SCIRXDA(=GPIO13), SCITXDA(=GPIO12)을 각각 easyDSP 포드의 RX, TX 에 연결합니다.



dual core 경우 (예를 들어 TMS320F28P65xD) easyDSP 를 CPU2 에도 사용하기 위해 CPU2 에 연결되는 easyDSP 는 SCI-B 에 연결합니다

SCI-B 의 GPIO 포트는 사용자가 설정 가능합니다만, easyDSP 가 제공하는 소스파일에는 GPIO86, GPIO87 을 사용하고 있습니다.

만약 SCI-B 의 GPIO 핀을 변경하고 싶은 경우, 적절하게 하드웨어 결선 및 easyDSP 소스파일(easyDSP_SCI_Init 함수 부분)을 직접 수정하시기 바랍니다.

기타 주의 사항 하기 참조 바랍니다.

- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- SCIRXD, SCITXD 는 해당 MCU 핀에 직접 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.
- /BOOT 핀은 2k 오옴 저항을 통해 GPIO72 에 연결
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함 (easyDSP pod 내부에 100 Ohm 직렬저항이 있기 때문)

또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

7.1.2.2 F2838x

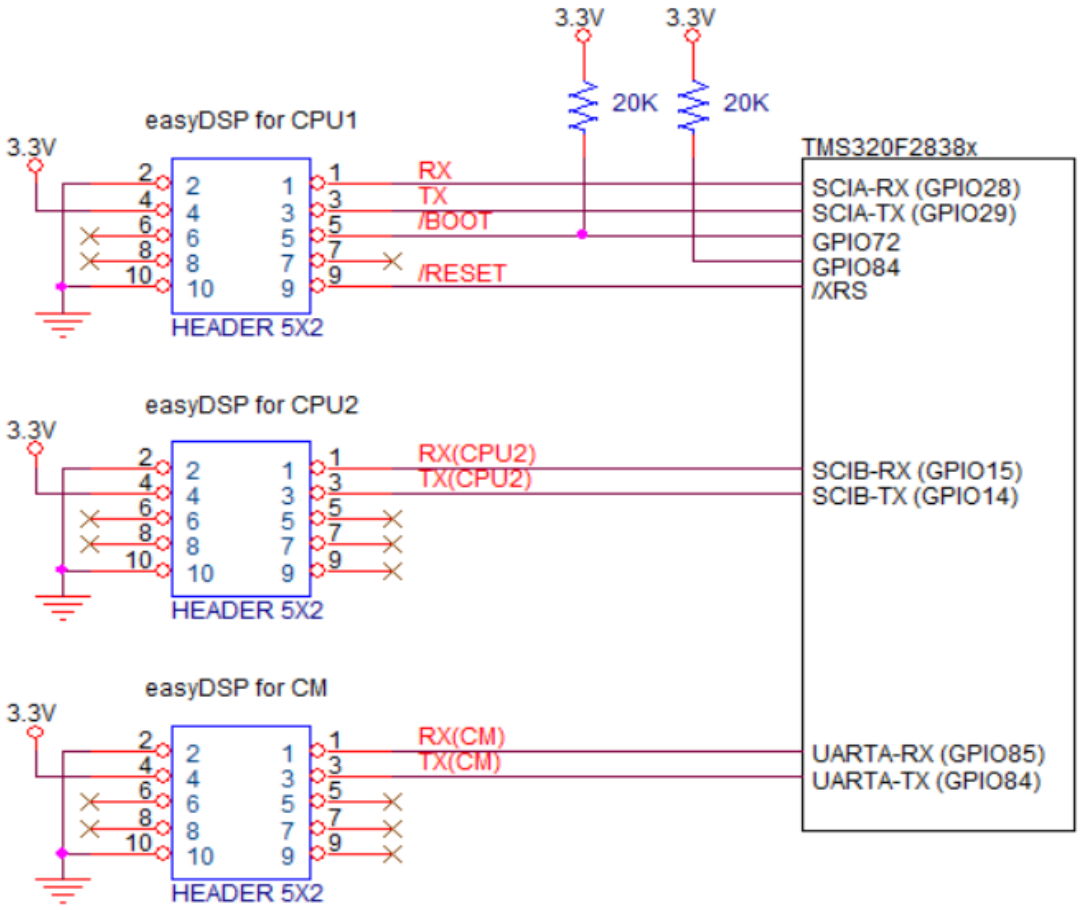
Delfino 시리즈 TMS320F2838xS(D) CPU1 는 디폴트로 리셋시 2 개 핀의 전위 상태에 따라 다음 표와 같이 부트 동작 모드를 결정합니다.

Boot Mode	GPIO72	GPIO84
Parallel IO	0	0
SCI /Wait Boot	0	1
CAN	1	0
Flash /USB	1	1

easyDSP 는 SCI 부트 (RAM 부트) / Flash 부트 모드만을 사용합니다. 따라서 하기 그림과 같은 결선을 권장합니다.

주의 사항 1 : SCIRXDA = GPIO28, SCITXDA = GPIO29 로 연결

주의 사항 2 : MCU 는 공장 출하 상태를 가정합니다. 사용자가 이후 customized 부트 모드를 사용할 경우, 본 페이지의 내용을 참조하셔서 적절히 수정하시기 바랍니다.



CPU1/CPU2/CM 을 모두 사용하기 위해서는 3 개의 easyDSP 가 필요합니다.

CPU1 에 연결되는 easyDSP 는 SCI-A 를 무조건 사용해야 하며 (GPIO28/29 고정),

CPU2 에 연결되는 easyDSP 는 SCI-B, SCI-C 또는 SCI-D 에 연결해서 사용 가능합니다만, 제공되는 easyDSP 소스파일은 SCI-B (GPIO14/15)를 사용하고 있습니다.

CM 에 연결되는 easyDSP 은 UART 를 사용해야 하며, 제공되는 easyDSP 소스파일은 GPIO84/85 를 사용하고 있습니다.

CPU2 및 CM 에서 연결되는 GPIO 를 변경하려면, 적절하게 하드웨어 및 easyDSP 제공 소스파일 (easyDSP_SCI_Init 함수 부분)을 직접 수정하셔야 합니다.

기타 주의 사항 하기 참조 바랍니다.

- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- RX, TX 는 해당 MCU 핀에 직접 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.
- /BOOT 핀은 2k 오옴 저항을 통해 GPIO72 에 연결
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함 (easyDSP pod 내부에 100 Ohm 직렬저항이 있기 때문).

사용자의 MCU 보드에서 이 4 개의 신호를 사용할 때, 별도의 풀업 또는 풀다운 처리를 할 경우 주의를 요합니다.

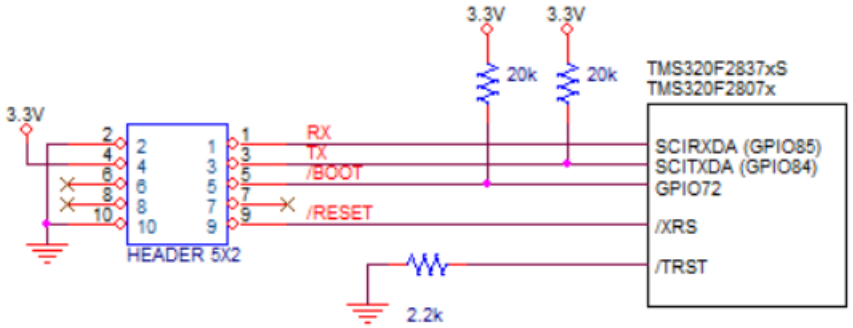
또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

7.1.2.3 F2837xS/2807x

Delfino 시리즈 TMS320F2837xS 및 Piccolo 시리즈 TMS320F2807x 모두는
리셋시 3 개 핀의 전위 상태에 따라 다음 표와 같이 부트 동작 모드를 결정합니다.

MODE	/TRST	GPIO72	GPIO84	Boot mode
Mode 0	0	0	0	Parallel I/O
Mode 1	0	0	1	SCI (RAM 부트)
Mode 2	0	1	0	Wait Boot Mode
Mode 3	0	1	1	Get Mode (공장 출하 설정은 플래쉬 부팅)
Mode 4-7	1	X	X	EMU Boot Mode (Emulator connected)

easyDSP 는 SCI 부트모드(RAM 부트) 및 GetMode 부트모드(Flash-ROM 실행 모드)만을 사용합니다. 따라서 하기 그림과 같은 결선을 권장합니다.
주의 사항) MCU 는 공장 출하 상태를 가정합니다. 사용자가 이후 설정을 변경하여, 부트 모드 핀이 변경되거나, Get Mode 의 기본 설정이 플래쉬 부팅이 아니거나 하게 되면 easyDSP 는 동작하지 않습니다.



- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- SCIRXDA = GPIO85, SCITXDA = GPIO84 로 연결
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.
- easyDSP 헤더와 MCU 간에 연결에 버퍼 IC 를 사용할 경우, 버퍼 IC 는 easyDSP 헤더에 직결하여, 각종 저항 연결이 DSP 에 직결될 수 있도록 함
- TX, RX 선은 MCU 의 해당 핀으로 직접 연결
- /BOOT 핀은 2k 오옴 저항을 통해 GPIO72 에 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함 (easyDSP pod 내부에 100 Ohm

직렬저항이 있기 때문)

사용자의 MCU 보드에서 이 4 개의 신호를 사용할 때, 별도의 풀업 또는 풀다운 처리를 할 경우 주의를 요합니다. 또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

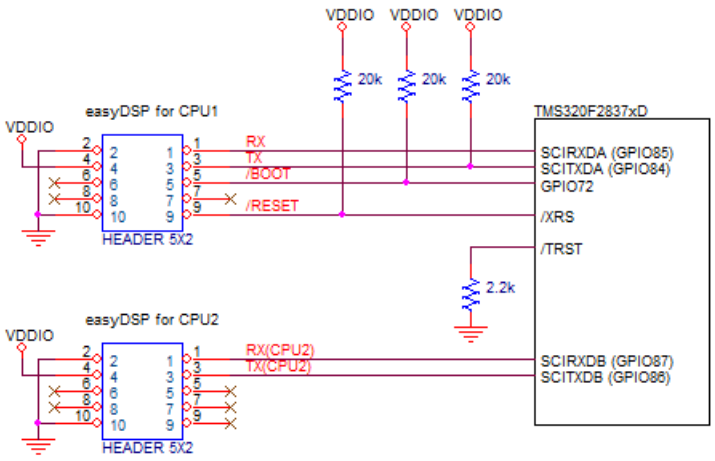
7.1.2.4 F2837xD

Delfino 시리즈 TMS320F2837xD 는
리셋시 3 개 핀의 전위 상태에 따라 다음 표와 같이 부트 동작 모드를 결정합니다.

MODE	/TRST	GPIO72	GPIO84	Boot mode
Mode 0	0	0	0	Parallel I/O
Mode 1	0	0	1	SCI (RAM 부트)
Mode 2	0	1	0	Wait Boot Mode
Mode 3	0	1	1	Get Mode (공장출하설정은 플래쉬 부팅)
Mode 4-7	1	X	X	EMU Boot Mode (Emulator connected)

easyDSP 는 SCI 부트모드(RAM 부트) 및 GetMode 부트모드(Flash-ROM 실행 모드)만을 사용합니다. 따라서 하기 그림과 같은 결선을 권장합니다.

주의 사항 1 : SCIRXDA = GPIO85, SCITXDA = GPIO84 로 연결
부득이 외부 메모리 인터페이스(External memory interface)를 사용하기 위해 GPIO84/85 를 사용할 수 없는 경우에는 '사용 포트 변경'을 참조하시기 바랍니다.
주의 사항 2 : MCU 는 공장 출하 상태를 가정합니다. 사용자가 이후 설정을 변경하여, 부트 모드 핀이 변경되거나, Get Mode 의 기본 설정이 플래쉬 부팅이 아니거나 하게 되면 easyDSP 는 동작하지 않습니다.



CPU1/CPU2 모두 사용하기 위해서는 2 개의 easyDSP 가 필요합니다.
CPU1 에 연결되는 easyDSP 는 DSP SCI-A 를 무조건 사용해야 하며 GPIO84/85 는 고정입니다.
CPU2 에 연결되는 easyDSP 는 SCI-B, SCI-C 또는 SCI-D 에 연결해서 사용 가능합니다만, easyDSP 는 SCI-B 를 사용하는 소스파일을 제공하고 있습니다.
만약 CPU2 의 GPIO 핀을 변경하고 싶은 경우, 적절하게 하드웨어 결선 및 easyDSP 소스파일(easyDSP_SCI_Init 함수 부분)을 직접 변경 바랍니다.

기타 주의 사항 하기 참조 바랍니다.

- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- SCIRXD, SCITXD 는 해당 MCU 핀에 직접 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.
- /BOOT 핀은 2k 오옴 저항을 통해 GPIO72 에 연결
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함 (easyDSP pod 내부에 100 Ohm 직렬저항이 있기 때문)

또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

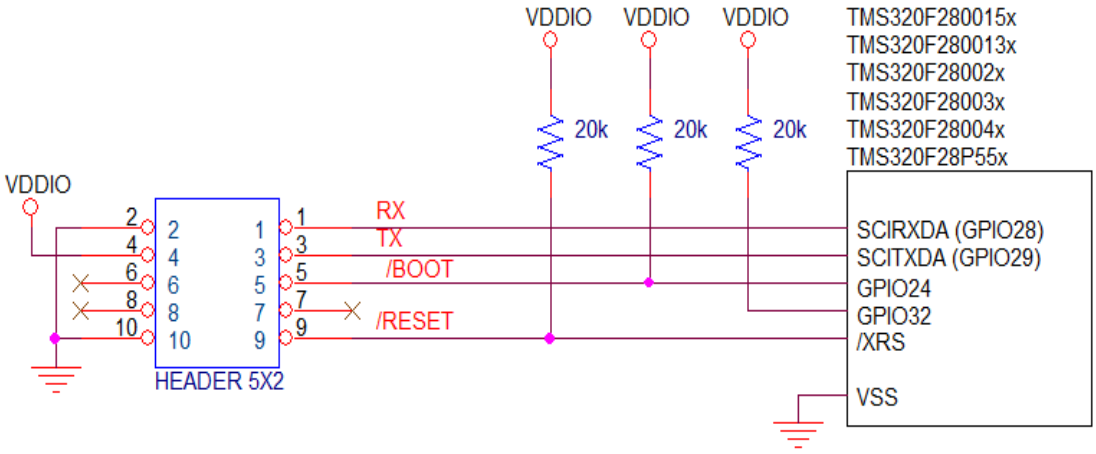
7.1.2.5 F28P55x/F28001x/28002x/28003x/28004x

공장 출하 기준 (OTP_BOOTPIN_CONFIG_KEY != 0x5A) 상태에서 디버거가 연결되지 않을 경우,
2 개 IO 핀의 전위 상태에 따라 다음 표와 같이 부트 동작 모드를 결정합니다.

MODE	GPIO24	GPIO32	Boot mode
Mode 0	0	0	Parallel I/O
Mode 1	0	1	SCI / Wait boot (RAM 부트)
Mode 2	1	0	CAN
Mode 3	1	1	Flash

주의 사항) MCU 는 공장 출하 상태를 가정합니다. 사용자가 이후 설정을 변경하여, 부트 모드 또는 보트 모드 핀이 변경될 경우 easyDSP 관련 설정도 변경되어야 합니다.

easyDSP 는 SCI 부트 모드(RAM 부트) 및 Flash 부트 모드만을 사용합니다. 따라서 하기 그림과 같은 결선을 권장합니다.



- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- SCIA_RX = GPIO28, SCIA_TX = GPIO29 로 연결
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함
- easyDSP 헤더와 MCU 간에 연결에 버퍼 IC 를 사용할 경우, 버퍼 IC 는 easyDSP 헤더에 직결하여, 각종 저항 연결이 DSP 에 직결될 수 있도록 함
- TX, RX 선은 MCU 의 해당 핀으로 직접 연결
- /BOOT 핀은 2k 오옴 저항을 통해 GPIO24 에 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함 (easyDSP pod 내부에 100 Ohm 직렬저항이 있기 때문)

사용자의 MCU 보드에서 이 4 개의 신호를 사용할 때, 별도의 풀업 또는 풀다운 처리를 할 경우 주의를 요합니다. 또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

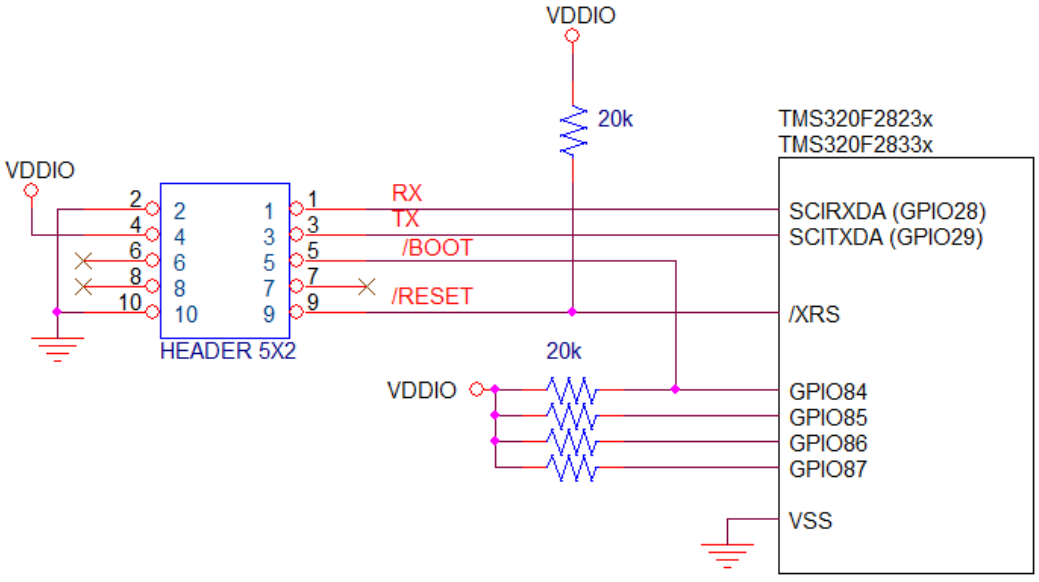
7.1.2.6 F2823x/2833x

TMS320F2823x/2833x 는 리셋시 4 개 포트의 전위상태에 따라 동작모드를 결정합니다.

MODE	GPIO87 XA15	GPIO86 XA14	GPIO85 XA13	GPIO84 XA12	Boot mode
F	1	1	1	1	Jump to Flash
E	1	1	1	0	SCI-A boot (RAM 부트)
D	1	1	0	1	SPI-A boot

C	1	1	0	0	I2C-A boot
B	1	0	1	1	eCAN-A boot
A	1	0	1	0	McBSP-A boot
9	1	0	0	1	Jump to XINTF x16
8	1	0	0	0	Jump to XINTF x32
7	0	1	1	1	Jump to OTP
6	0	1	1	0	Parallel GPIO I/O boot
5	0	1	0	1	Parallel XINTF boot
4	0	1	0	0	Jump to SARAM
3	0	0	1	1	Branch to check boot mode
2	0	0	1	0	Branch to Flash, skip ADC calibration
1	0	0	0	1	Branch to SARAM, skip ADC calibration
0	0	0	0	0	Branch to SCI, skip ADC calibration

easyDSP 에서 Flash-ROM 실행 모드와 SCI 부트(RAM 부트)만을 사용하므로 3 개의 핀(GPIO87, 86, 85)의 상태는 고정하고 1 개의 핀(GPIO84) 상태만 변경 사용하여 동작 모드를 결정합니다.
따라서 하기 그림과 같은 결선을 권장합니다.



- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- TX, RX 선은 MCU 의 해당 핀으로 직접 연결

- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.
- easyDSP 헤더와 MCU 간에 연결에 버퍼 IC 를 사용할 경우, 버퍼 IC 는 easyDSP 헤더에 직결하여, 각종 저항 연결이 DSP 에 직결될 수 있도록 함
- /BOOT 핀은 2k 오옴 저항을 통해 GPIO84 (또는 GPIO85) 에 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함 (easyDSP pod 내부에 100 Ohm 직렬저항이 있기 때문)

사용자의 MCU 보드에서 이 4 개의 신호를 사용할 때, 별도의 풀업 또는 풀다운 처리를 할 경우 주의를 요합니다.
또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

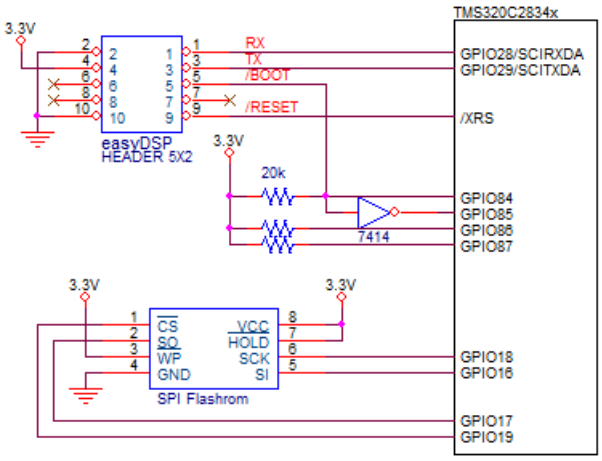
7.1.2.7 C2834x

TMS320C2834x 는 리셋시 4 개 포트의 전위상태에 따라 부트 동작모드를 결정합니다.

MODE	GPIO87 XA15	GPIO86 XA14	GPIO85 XA13	GPIO84 XA12	부팅 모드
E	1	1	1	0	SCI-A boot (RAM 부팅시 사용)
D	1	1	0	1	SPI-A boot (외부 flashrom 부팅용으로 권장함)

easyDSP Pod 는 모드 E 번을 사용하여 램 부팅을 수행합니다.
램 부팅시에는 easyDSP pod 의 /Boot 신호 및 /Reset 신호가 동시에 활성화되며,
사용자 메뉴(DSP 메뉴> Reset DSP 메뉴)에 의한 리셋시에는 /Reset 신호만 활성화되므로, 경우에 따라 하기와 같이 결선하여 주십시요.

상기 테이블의 파란색 칸은 플래쉬롬 부팅을 위한 easyDSP 의 추천 방식입니다. 하드웨어 준비는 사용자의 몫입니다.



(easyDSP Pod 결선법 : 시리얼부팅은 SCI-A boot 모드 사용. 리셋시에는 SPI-A 부팅사용)

기타 주의사항은 아래와 같습니다.

- SPI-A 는 easyDSP 전용으로 사용되므로 사용자 용도로 사용될 수 없습니다.
- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- TX, RX 선은 MCU 의 해당 핀으로 직접 연결
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.
- easyDSP 헤더와 MCU 간에 연결에 버퍼 IC 를 사용할 경우, 버퍼 IC 는 easyDSP 헤더에 직결하여, 각종 저항 연결이 DSP 에 직결될 수 있도록 함
- /BOOT 핀은 2k 오옴 저항을 통해 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함 (easyDSP pod 내부에 100 Ohm 직렬저항이 있기 때문)
- 노이즈에 의한 리셋을 방지하기 위해, 리셋 회로부에 적절한 필터 사용 가능

주의사항 !!

easyDSP 의 'MCU'메뉴의 'Reset MCU'를 선택하여 MCU 에 리셋을 거는 경우에는, /RESET 신호만이 활성화됩니다. 이때에는 /BOOT 신호는 비활성화됩니다. 따라서 상기와 같이 결선한 후, 'MCU'메뉴의 'Reset MCU'를 선택하여 리셋을 걸면, MCU 는 모드 D(SPI-A boot)로 진입하게 됩니다. **따라서 "SPI-A"모드로 부팅이 준비되어 있지 않을 경우에는 'MCU'메뉴의 'Reset MCU'를 사용하여 MCU 를 리셋시키지 마십시요.**

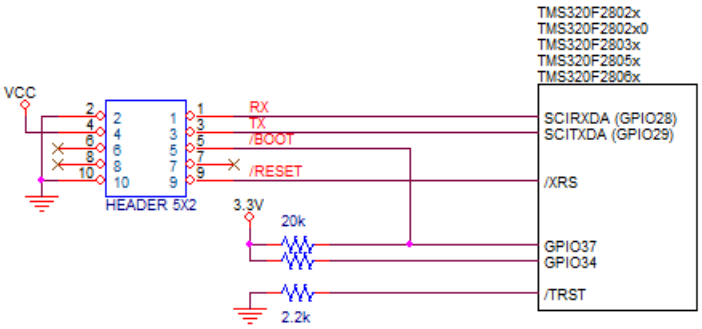
7.1.2.8 F2802x/2802x0/2803x/2805x/2806x

Piccolo 시리즈 TMS320F2802x/2802x0/2803x/2805x/2806x 는 리셋시 3 개 포트의 전위상태에 따라 다음 표와 같이 부트동작모드를 결정합니다.

MODE	GPIO37 TDO	GPIO34 CMP2OUT	/TRST	Boot mode
Mode EMU	X	X	1	Emulation Boot

Mode 0	0	0	0	Parallel I/O
Mode 1	0	1	0	SCI (RAM 부트)
Mode 2	1	0	0	Wait
Mode 3	1	1	0	GetMode

easyDSP 는 SCI 부트모드(RAM 부트) 및 GetMode 부트모드(Flash-ROM 실행 모드)만을 사용하므로, 에뮬레이터를 연결하지 않은 경우에는 (즉 /TRST = 0), < /FONT > < /FONT > GPIO34 핀의 상태는 1 로 고정하고 GPIO37 핀의 상태만 변경하여 동작 모드를 결정합니다. 따라서 하기 그림과 같은 결선을 권장합니다.
단, 에뮬레이터를 연결한 경우에는 부트 모드가 하기 결선과 관계없이 특정 주소의 메모리값에 의존됩니다. 이에 대해서는 TI 의 자료를 참조하십시요.



- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- SCIRXDA = GPIO28, SCITXDA = GPIO29 로 연결
- TX, RX 선은 MCU 의 해당 핀으로 직접 연결
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.
- easyDSP 헤더와 MCU 간에 연결에 버퍼 IC 를 사용할 경우, 버퍼 IC 는 easyDSP 헤더에 직결하여, 각종 저항 연결이 DSP 에 직결될 수 있도록 함
- /BOOT 핀은 2k 오옴 저항을 통해 GPIO37 에 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함 (easyDSP pod 내부에 100 Ohm 직렬저항이 있기 때문)

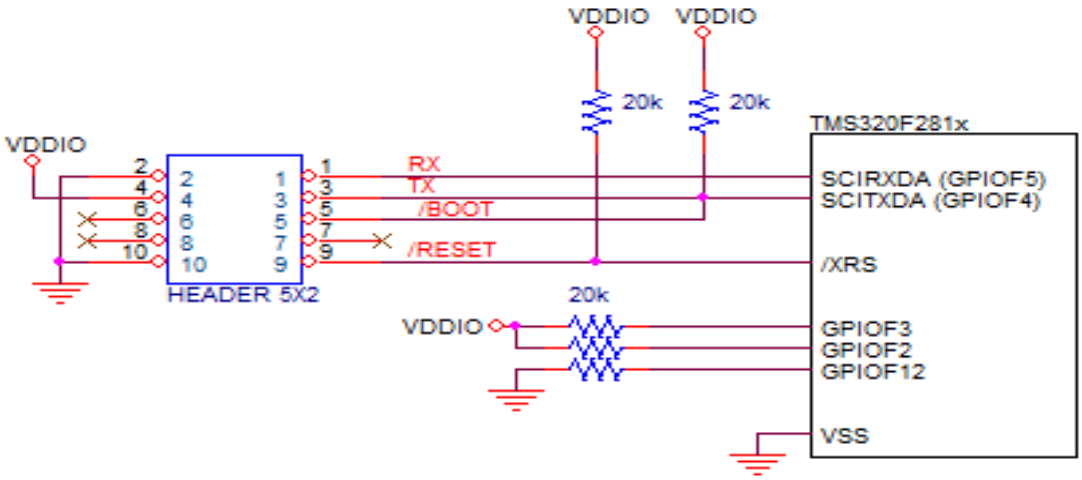
사용자의 MCU 보드에서 이 4 개의 신호를 사용할 때, 별도의 풀업 또는 풀다운 처리를 할 경우 주의를 요합니다. 또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

7.1.2.9 F281x

TMS320F281x 는 리셋시 4 개 포트의 전위상태에 따라 동작모드를 결정합니다.

GPIOF4(SCITXDA)	GPIOF12(MDXA)	GPIOF3(SPISTEA)	GPIOF2(SPICLKA)	부트 모드
1	x	x	x	**FLASH(0x3F7FF6) 실행**
0	1	x	x	SPI 부트 실행
0	0	1	1	**SCI 부트(SCI-A) 실행** (RAM 부트)
0	0	1	0	H0 SARAM(0x3F8000) 실행
0	0	0	1	OTP (0x3D7800) 실행

easyDSP 는 Flash-ROM 실행 모드와 SCI 부트(RAM 부트)만을 사용하므로 3 개의 핀(MDXA, SPISTEA, SPICLK)의 상태는 고정하고 1 개의 핀(SCITXDA) 상태만 변경 사용하여 동작 모드를 결정합니다. 따라서 하기 그림과 같은 결선을 권장합니다.



- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- TX, RX 선은 MCU 의 해당 핀으로 직접 연결
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.
- easyDSP 헤더와 MCU 간에 연결에 버퍼 IC 를 사용할 경우, 버퍼 IC 는 easyDSP 헤더에 직결하여, 각종 저항 연결이 DSP 에 직결될 수 있도록 함
- /BOOT 핀은 2k 오옴 저항을 통해 SCITXDA 에 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야함 (easyDSP pod 내부에 100Ohm 직렬저항이 있기 때문)

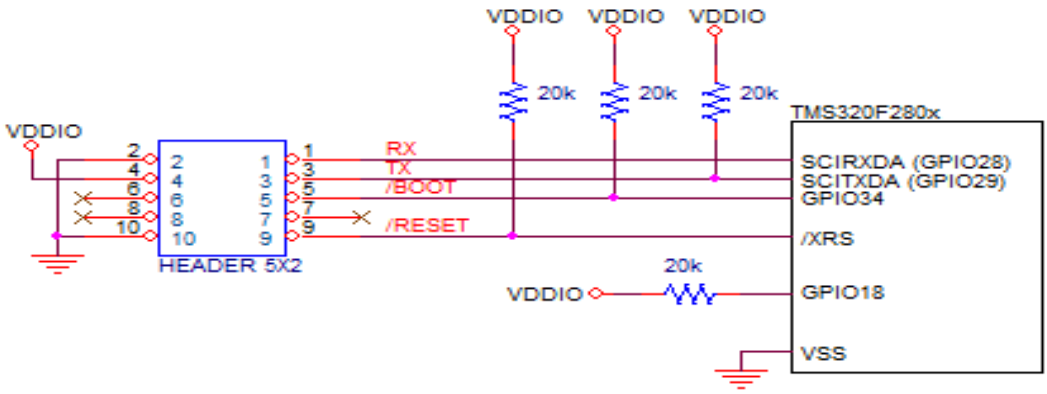
사용자의 MCU 보드에서 이 4 개의 신호를 사용할 때, 별도의 풀업 또는 풀다운 처리를 할 경우 주의를 요합니다. 또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

7.1.2.10 F280x

TMS320F280x 는 리셋시 3 개 포트의 전위상태에 따라 동작모드를 결정합니다.

부트모드	GPIO18 SPICLKA SCITXB	GPIO29 SCITXDA	GPIO34
Jump to Flash 0x3F 7FF6	1	1	1
Call SCI-A boot loader (RAM 부트)	1	1	0
Call SPI-A boot loader	1	0	1
Call I2C-A boot loader	1	0	0
Call eCAN-A boot loader	0	1	1
Jump to M0 SARAM 0x00 0000	0	1	0
Jump to OPT	0	0	1
Parallel GPIO Loader	0	0	0

easyDSP 는 Flash-ROM 실행 모드와 SCI-A 부트(RAM 부트)만을 사용하므로 2 개의 핀(GPIO18, GPIO29)의 상태는 고정하고 1 개의 핀(GPIO34) 상태만 변경하여 동작 모드를 결정합니다. 따라서 하기 그림과 같은 결선을 권장합니다.



- easyDSP cable 이 연결되는 5x2 header 의 전원핀(#4)은 3.3V 연결
- TX, RX 선은 MCU 의 해당 핀으로 직접 연결
- easyDSP /RESET 신호와 MCU /XRS 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 /XRS 에 전달해야 함.

- easyDSP 헤더와 MCU 간에 연결에 버퍼 IC 를 사용할 경우, 버퍼 IC 는 easyDSP 헤더에 직결하여, 각종 저항 연결이 DSP 에 직결될 수 있도록 함
- /BOOT 핀은 2k 오옴 저항을 통해 GPIO34 에 연결
- /Reset 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/Reset 핀의 Low 상태 유지 기간은 약 500msec)
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야함 (easyDSP pod 내부에 100Ohm 직렬저항이 있기 때문)

사용자의 MCU 보드에서 이 4 개의 신호를 사용할 때, 별도의 풀업 또는 풀다운 처리를 할 경우 주의를 요합니다. 또한 리셋핀 사용시, 여러 원하지 않는 상황에 의해, 의도되지 않은 리셋이 발생될 수 있습니다. 이를 방지하기 위해 적절한 필터를 사용하는 등의 주의를 기울여 주십시오.

7.1.3 SCI-A 가 아닌 포트 사용

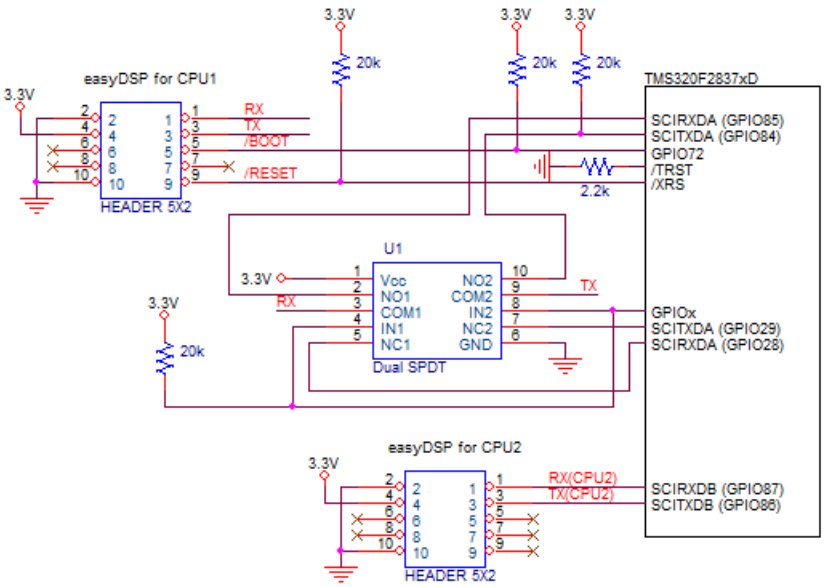
앞서 "C28x 보드 세팅"에서 권장한 포트를 사용할 경우에만, easyDSP 를 이용하여 부팅 및 모니터링을 모두 수행할 수 있습니다. 이는 MCU 가 SCI-A 로만 부팅 동작을 지원하기 때문입니다. 만약 SCI B 포트에 easyDSP 를 연결한다면, 부팅이 지원되지 못하게 됩니다.
특별한 사유로 SCI-A 포트가 아닌 다른 포트를 easyDSP 에 연결해야 할 경우, 하기의 방법을 통해 부팅 및 모니터링 모두 동작 가능하게 할 수 있습니다. 하기는 TMS320F28377D 경우를 설명하지만, 다른 MCU 에도 동일한 방법이 적용 가능합니다.

easyDSP 를 TMS320F28377D GPIO85, GPIO84 가 아닌 다른 GPIO 포트에 연결하여 사용하고자 할 경우 제안 드리는 방법 :

부트롬에 의한 MCU 부팅은 GPIO85/GPIO84(SCI-A)에 연결할 경우에만 가능하며, 이로써 램 부팅 및 플래시 프로그래밍이 수행 가능합니다. 따라서 easyDSP 를 다른 포트에 연결 사용할 경우 램 부팅 및 플래시 프로그래밍은 불가능하며, 모니터링만 가능하게 됩니다.
easyDSP 를 GPIO85/GPIO84 포트에 연결하여 사용할 수 없는 경우라면 (예를 들어 EMIF 사용), 적절한 방법으로 상기 문제를 회피할 수 있습니다.
일단 easyDSP 를 GPIO85, GPIO84 에 연결하여 부팅한 후, 부팅 이후 사용자 프로그램에서 easyDSP 의 결선을 다른 포트로 변경하는 것입니다. 결선을 바꾸기 위해서는 부가적인 하드웨어가 필요하며 하기를 참조하세요. Dual SPDT 스위치(NLAS4684 (Onsemi 사),TS3A24159 (TI 사)) 또는 FPGA 를 사용합니다.

http://www.onsemi.com/pub_link/Collateral/NLAS4684-D.PDF

http://www.ti.com/lit/ds/symlink/ts3a24159.pdf



결선을 바꾸기 위해 부가적인 GPIOx 가 할당되었습니다. 사용하지 않는 GPIO 중 하나를 본 용도로 할당해 주십시오. 리셋 이후 부팅 동작 동안은 GPIOx 가 입력단자(리셋 초기값)이므로 외부 풀업 저항에 의해 dual SPDT 의 IN 단자에 1 이 입력되어 easyDSP 결선은 GPIO85/84 에 연결됩니다. 이후 부팅을 완료한 후 사용자 프로그램에서 GPIOx 를 출력단자로 설정하시고 값을 0 으로 변경하여 easyDSP 결선을 다른 포트(회로에서는 GPIO28/29)로 할당하는 원리입니다.

결선을 바꾸는 시점 및 방법은 CPU1 의 easyDSP_SCI_Init()에서 구현합니다. 즉, easyDSP_SCI_Init()의 기존 해당 부위를 적절히 수정합니다.

원래 CPU1 의 easyDSP_SCI_Init()의 하기 부분을

```
//////////////////////////////////////////
// SCI-A GPIO setting : SCIRXDA = GPIO 85, SCITXDA = GPIO84
//////////////////////////////////////////
GPIO_SetupPinMux(84, GPIO_MUX_CPU1, 5);
GPIO_SetupPinMux(85, GPIO_MUX_CPU1, 5);
GPIO_SetupPinOptions(84, GPIO_OUTPUT, GPIO_ASYNC);
GPIO_SetupPinOptions(85, GPIO_INPUT, GPIO_ASYNC);
EALLOW;
GpioCtrlRegs.GPCPUD.bit.GPIO85 = 0;
GpioCtrlRegs.GPCPUD.bit.GPIO84 = 0;
EDIS;
```

아래와 같이 변경합니다 (GPIO28/29 그리고 GPIO31 을 사용할 경우)

```
//////////////////////////////////////////
// SCI-A GPIO setting : SCIRXDA = GPIO 28, SCITXDA = GPIO29
```

```
//////////////////////////////////////////
GPIO_SetupPinMux(29, GPIO_MUX_CPU1, 1);
GPIO_SetupPinMux(28, GPIO_MUX_CPU1, 1);
GPIO_SetupPinOptions(29, GPIO_OUTPUT, GPIO_ASYNC);
GPIO_SetupPinOptions(28, GPIO_INPUT, GPIO_ASYNC);
EALLOW;
GpioCtrlRegs.GPCPUD.bit.GPIO28 = 0;
GpioCtrlRegs.GPCPUD.bit.GPIO29 = 0;
EDIS;
// easyDSP 포트를 GPIO28/29 에 연결 (GPIO31 을 사용하여)
GPIO_SetupPinMux(31, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(31, GPIO_OUTPUT, GPIO_PUSHPULL);
GPIO_WritePin(31, 0);
```

마지막으로 GPIO72 핀도 EMIF 에서 사용되므로 easyDSP 포드의 #5 핀 상태도 고려해야 합니다. 해당 핀은 유사 오픈 컬렉터 출력 타입이므로, 부팅시에만 Low 값이 출력 된 후 이후에는 오픈 상태가 유지됩니다. 따라서 해당 핀을 EMIF 로 사용하는 데에 추가적인 고려 사항은 없습니다. 단, GPIO72 핀이 easyDSP 포드에 연결되어 있으므로, MCU 동작시 easyDSP 포드를 찰탈착하는 것은 의도치 않은 노이즈 신호를 만들 수 있으므로 금지합니다. NEW!

Get mode 사용의 제한점 :

또 다른 방식으로는 Get Mode 의 SCI BOOT 1 을 사용할 수 있습니다. 단, 이 경우 Zx-BOOTCTRL 레지스터를 변경하여야 합니다. Zx-BOOTCTRL 레지스터는 OTP 영역이므로 한번 고정하면 바꿀 수 없으며, Flash booting 이 불가능하게 됩니다.

7.1.4 C28x 주의사항

C28x 주의 사항

---

* 프로그램 코드 사이즈가 클 때 램 부팅에 실패한다면 ?

이러한 상황은 특히 TMS320C2834x 시리즈에서 발생됩니다. 프로그램 코드 사이즈가 커지면서 c_int00 에서 cinit 섹션 변수들을 초기화하는 시간이 오래 걸리게 됩니다. 이때 일정 시간이 초과하게 되면 와치독이 작동하여 리셋이 걸리게 되며, 리셋후 MCU 는 램 부팅이 아닌 다른 부팅 모드(보통 플래시 부팅)로 부팅하게 되어 결국 램 부팅은 실패하게 됩니다. 이를 예방하기 위해서는 c_int00 에 진입하기 전에 와치독을 비활성화시키는 code_start (TI 예제 파일에서 제공)를 entry point 로 지정하여 주십시오. 링커 옵션에서 -ecode_start 를 사용하세요.

* XDS100 을 같이 사용할 때는

TI 사의 에뮬레이터 XDS100v1 을 CCS v3.3 에서 사용하실 경우, easyDSP 와 같이 사용할 수 없습니다. 이는, CCS v3.3 에서 XDS100v1 를 사용할 경우, FTDI 사의 USB 제어 칩을 다중으로 지원하지 않기 때문입니다.
XDS100v1(또는 XDS100v2)를 CCS v4 이후 버전에서 사용하실 경우에는 easyDSP 와 같이 사용될 수 있습니다.

* easyDSP 가 지정한 포트를 사용하지 않고 다른 포트를 사용할 경우

예를 들어, SCI 통신하는 F28335 의 경우, easyDSP 는 SCIRXDA, SCITXDA 로 각각 GPIO28 번, GPIO29 번을 사용하도록 지정하고 있습니다. 만약 사용자가 SCIRXDA, SCITXDA 로 각각 GPIO36 번, GPIO35 번을 사용하게 되면, 일반적인 모니터링 통신은 가능하지만, SCI 부팅이 안되어 램부팅 및 플래시 프로그래밍이 동작하지 않게 됩니다. F28335 부트롬에 SCI 부팅은 GPIO28 번, GPIO29 번 포트만을 통해서 이루어지도록 지정되어 있기 때문입니다.

* easyDSP 통신 도중에 MCU 를 리셋시키면?

리셋 이후 MCU 가 플래쉬롬 부팅을 한다면 문제가 없습니다.
만약 기본 부팅모드가 램 부팅으로 되어 있다면 매우 적은 확률이지만 MCU 가 폭주할 위험이 있습니다. 왜냐하면, MCU 는 램 부팅모드로 진입되고, 이때 easyDSP 가 통신을 위하여 내보내는 신호를 MCU 는 램 부팅을 위한 데이타로 인식하게 됩니다. 램 부팅을 위한 데이타에도 준수되어야 할 형식이 있으므로, 통신을 위한 데이타가 부팅을 위한 데이타로 인식될 확률은 매우 적습니다만.

7.2 STM32

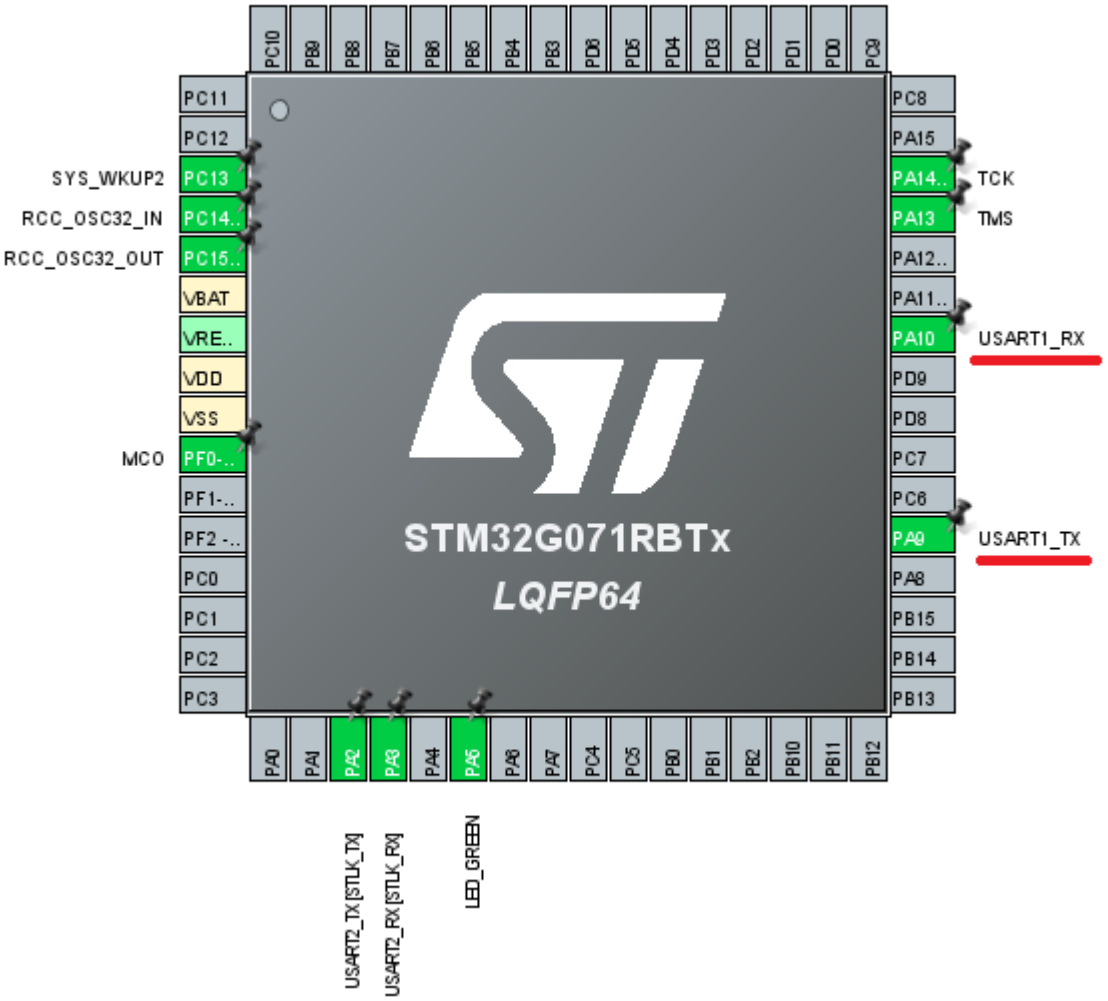
7.2.1 STM32 프로그래밍

STEP 1 : USART 채널 및 관련 설정

STM32CubeMX 기준으로 설명 드립니다.

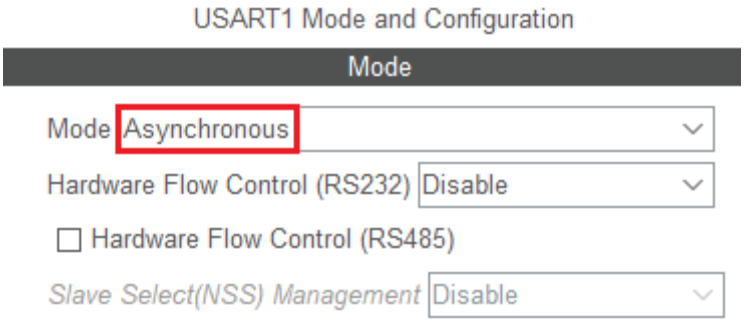
설명	STM32CubeMX

easyDSP 가 연결될 USART 채널을 선정합니다. UART 채널은 사용할 수 없습니다. STM32 하드웨어 설정>STEP1 을 참조하시기 바랍니다.



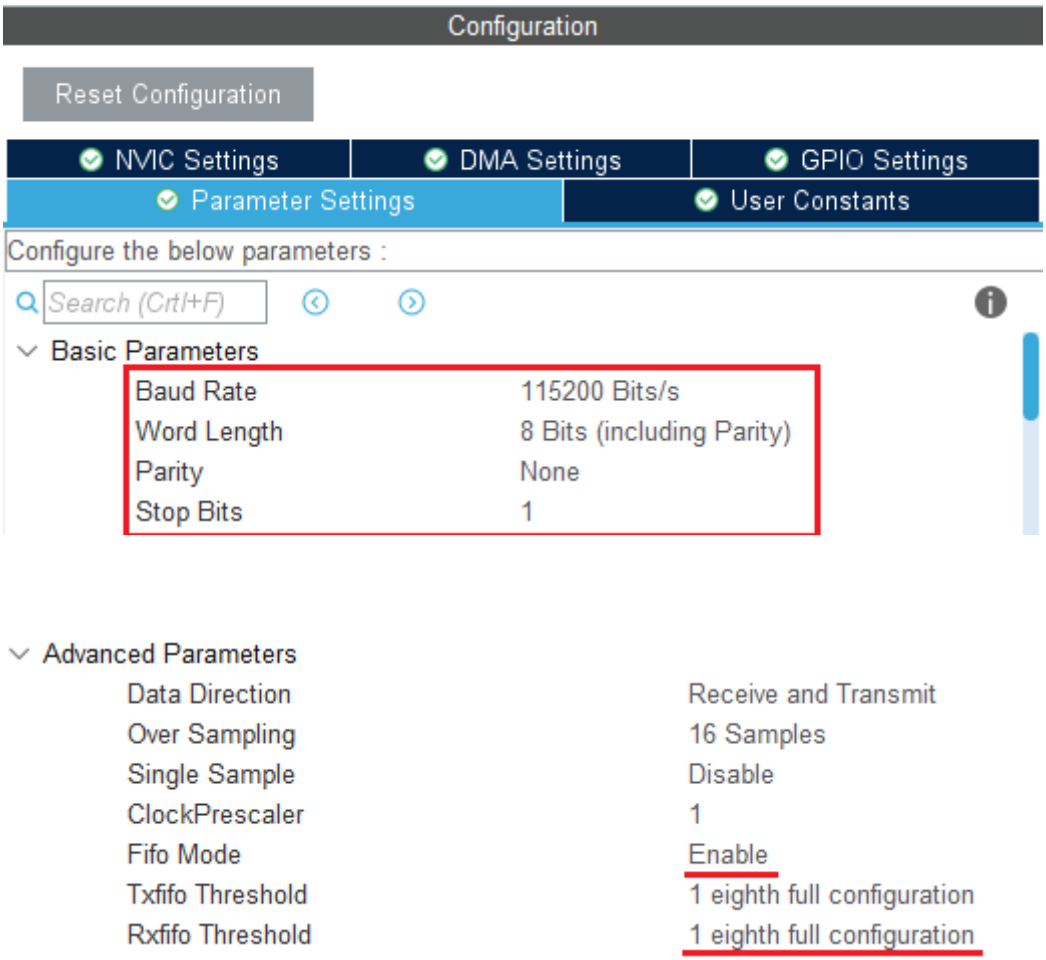
본 예제에서는 USART1 을 선정하였습니다.

connectivity 카테고리의 선택된 USART 채널로 이동합니다.먼저 USART 채널을 Asynchronous mode 로 활성화 시킵니다.



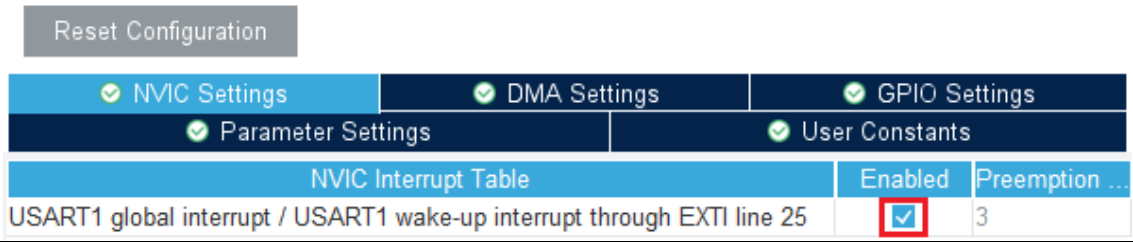
해당 USART 채널의 통신 파라미터를 설정합니다.

8 bits, no parity, 1 stop bits 은 반드시 지켜져야 하며 , 보드레이트는 선택 가능합니다 (115200 bps 추천).

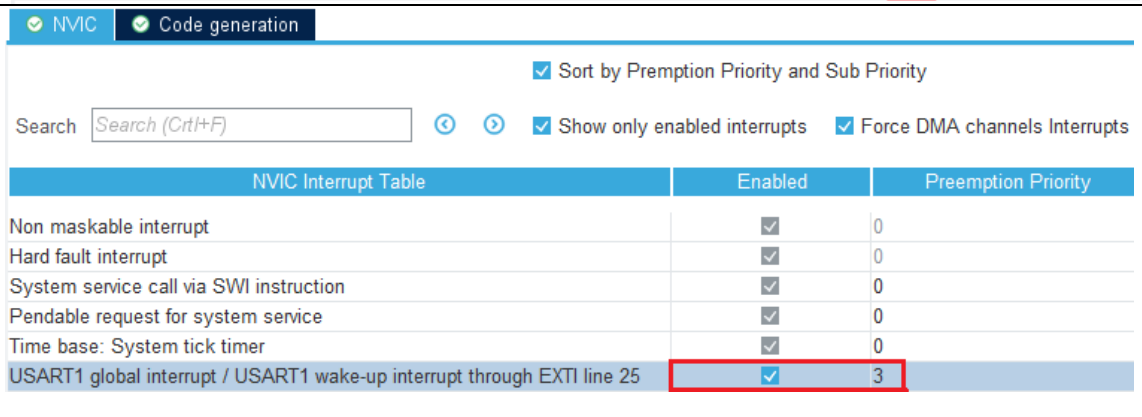


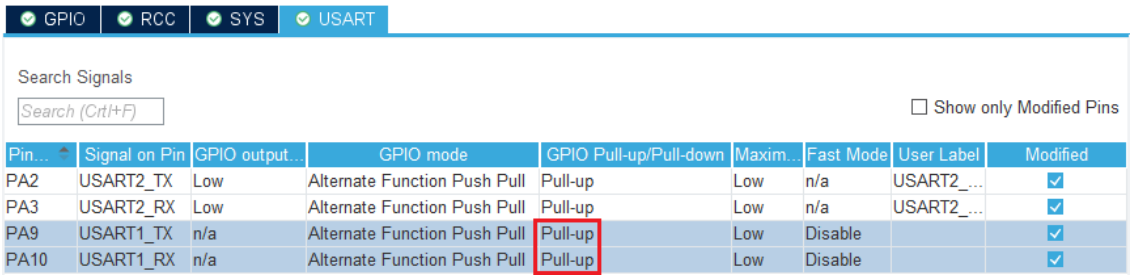
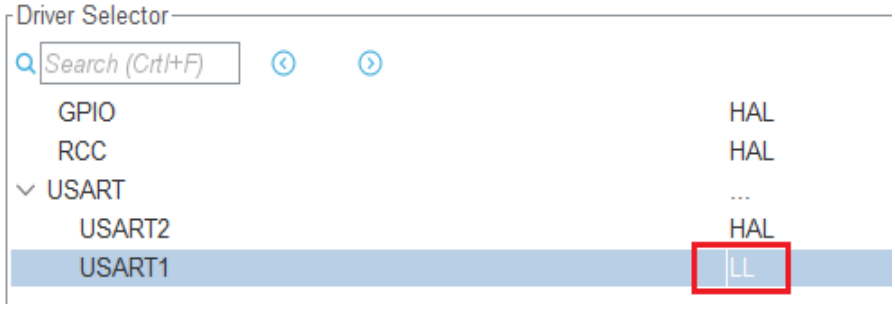
만약 USART 에 8 레벨 이상의 FIFO 기능이 있는 MCU 라면 FIFO 기능을 활성화시키고, Rxfifo Threshold 는 1 eight full configuration 으로 설정합니다.
주의 : 이 경우 easyStm32LL.c 버전 10.5 이상이 사용되어야 함.

또한 해당 USART 채널의 인터럽트를 활성화 시켜 주십시요.



System Core 카테고리의 NVIC 항으로 이동하여, 해당 USART 인터럽트의 순위를 최하위로 조정합니다. 즉, 가장 높은 숫자로 설정.



System Core 카테고리의 GPIO 항으로 이동하여, 해당 USART 의 GPIO Pull-up/Pull-down 설정을 Pull-up 으로 지정합니다.	
Project Manager > Advanced Settings > Driver Selector 항에서 사용할 USART 채널에 대해서 **LL 사용을 지정합니다 .**	

STEP 2 : LL 사용한 USART 인터럽트 서비스 루틴 관련 처리

easyDSP 는 MCU 의 USART 통신으로 PC 와 MCU 간 통신을 구현합니다. 따라서 사용자의 MCU 프로그램은 easyDSP 가 제공하는 USART 인터럽트 처리 루틴 (Interrupt Service Routine,이하 ISR)을 포함하여야 합니다 .
HAL 기반 통신 대비 리소스 소모가 적어서 본 LL 기반 통신을 사용하고 있습니다.
LL 기반 소스 파일은 아래와 같으며, easyDSP 인스톨된 폴더의 'Source > STM32' 폴더에 위치합니다.
easyStm32LL_v11.4.c
easyStm32LL_v11.4.h

하기 단계별 작업 참조하세요. 멀티 코어 사용시 추가되는 옵션 설정은 여기를 참조하시기 바랍니다.
작업이 완료되면 easyDSP 는 변수 모니터링 및 변경을 지원할 수 있습니다.

단계	소스 코드 예제

사용 MCU 종류에 맞춰 ,
easyStm32LL.h
파일을 수정하세요 .

참고로 easyStm32LL.c 파일은
수정하지 않습니다.

```
////////////////////////////////////////////////
// Select target MCU series :
// Define 1 to target MCU. 0 to all others.
////////////////////////////////////////////////
#define STM32C0XX           0
#define STM32F0XX           0
#define STM32F1XX           0
#define STM32F2XX           0
#define STM32F3XX           0
#define STM32F4XX           0
#define STM32F7XX           0
#define STM32G0XX           1
#define STM32G4XX           0
#define STM32H5XX           0
#define STM32H7XX           0
#define STM32L0XX           0
#define STM32L1XX           0
#define STM32L4XX           0
#define STM32L5XX           0
#define STM32U5XX           0
#define STM32WBXX           0
#define STM32WBAXX          0
#define STM32WLXX           0
```

예제는 STM32G0 시리즈를 사용할 경우의 헤더 파일 예제입니다.

이후 main.c 의 앞부분에
헤더파일을 추가시키고,

main()
함수에서 MX_USARTx_UART_Init()가
호출된 이후,
easyDSP_init(USARTz)함수를
호출합니다.

단, z = 선택된 USART 채널.
예제에서는 USART1 를 사용하기
위해 USART1 를 입력하고 있습니다.

```
/* USER CODE BEGIN Includes */
#include "easyStm32LL_vx.y.h"   // x.y = version x.y
/* USER CODE END Includes */

int main(void)
{
    .
    .
    .
    .
    .
    .
    MX_USART1_UART_Init();
    .
    .
    .

    /* USER CODE BEGIN 2 */
    easyDSP_init(USART1);
    /* USER CODE END 2 */

    while (1)
    {

    }
}
```

ISR 이 정의된 파일 (G0 시리즈라면 stm32g0xx_it.c) 앞부분에 헤더파일을 추가시키고,

해당 USART 채널의 ISR 함수에 (본 예제에서는 USART1) ez_USARTx_IRQHandler()를 호출합니다.

```
/* USER CODE BEGIN Includes */
#include "easyStm32LL vx.y.h"    // x.y = version x.y
/* USER CODE END Includes */

void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQn 0 */
    ez_USARTx_IRQHandler();
    /* USER CODE END USART1_IRQn 0 */
    /* USER CODE BEGIN USART1_IRQn 1 */

    /* USER CODE END USART1_IRQn 1 */
}
```

STEP 3 : 듀얼 코어

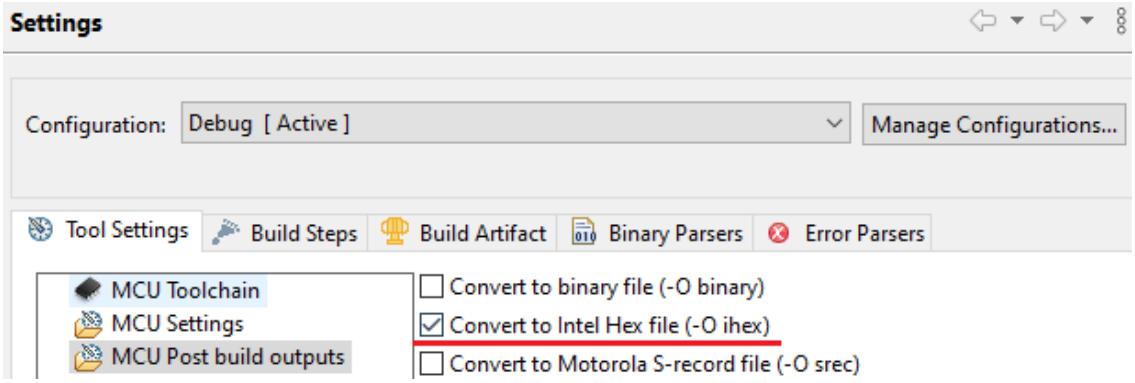
듀얼 코어 MCU 의 경우 플래시 동일 페이지 안에 다른 코어의 프로그램이 동시에 존재하지 말아야 합니다.

STEP 4 : IDE 설정

1. 매 컴파일마다 hex 파일(인텔 형식)이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 해주세요. hex 파일은 램 부팅 또는 플래시 프로그래밍할 때 사용됩니다.

Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.

예를 들어, Stm32CubeIde 의 경우라면 아래와 같이 설정합니다.



2. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다. 이를 위해 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

2.1. 어셈블러/컴파일러/링커가 debug information 을 반드시 포함하도록 설정

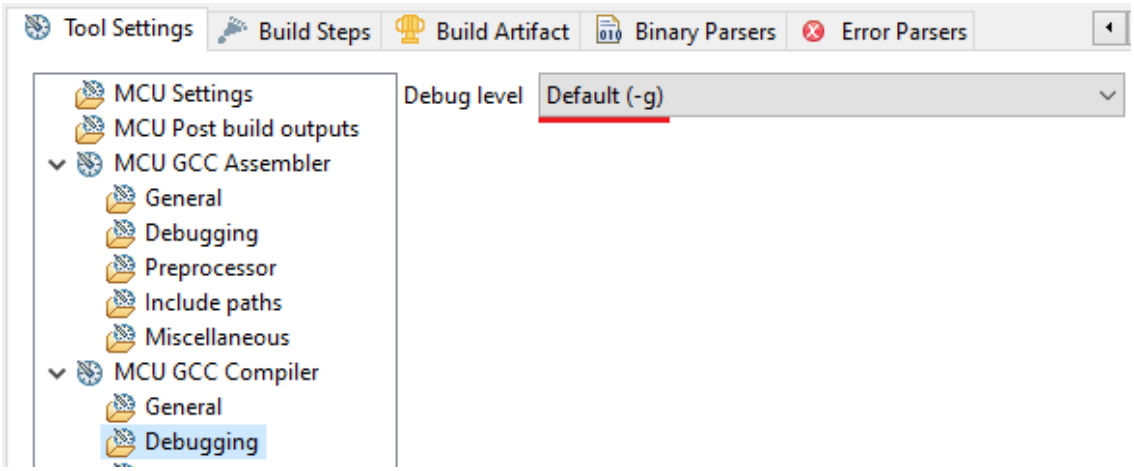
예를 들어, Stm32CubeIde 의 경우라면 컴파일러 옵션에 -g 옵션 사용

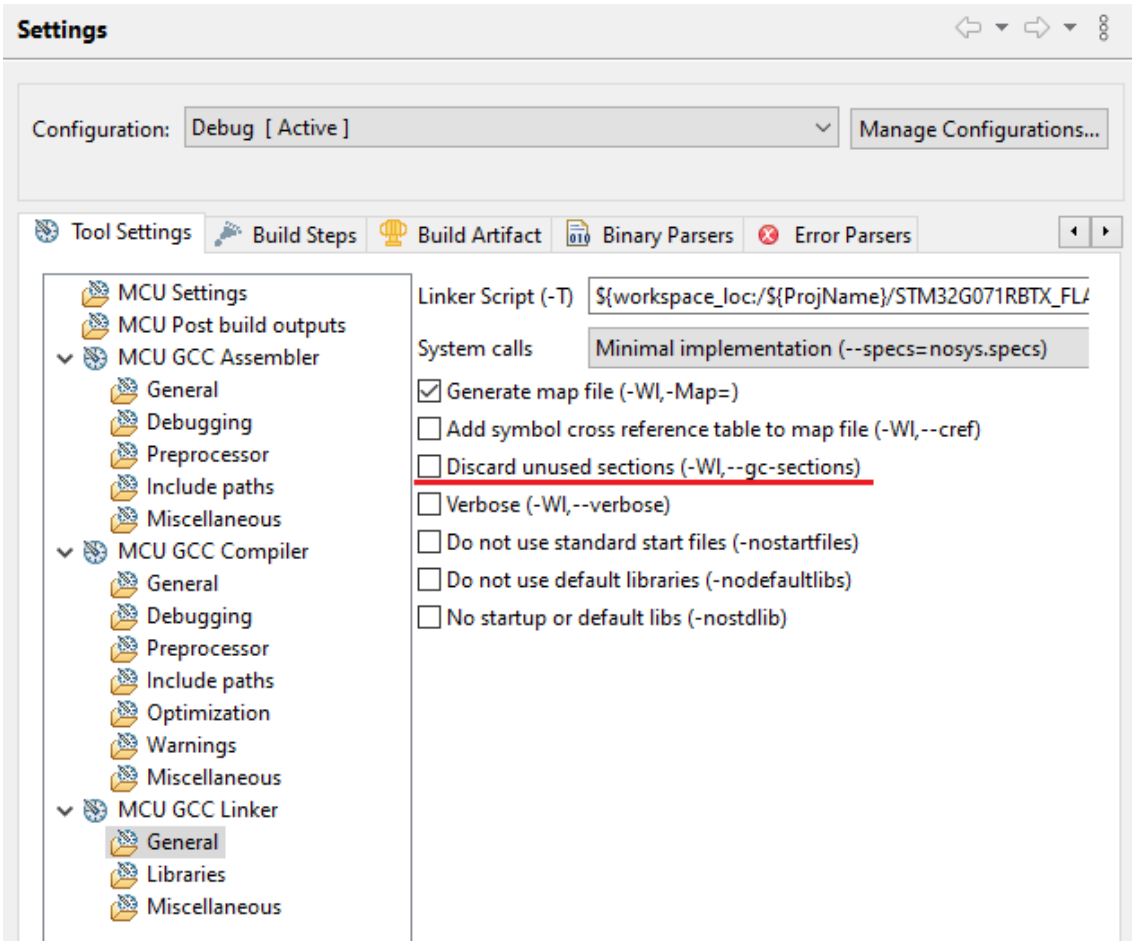
2.2. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 포함되게 하기 위해서라면

예를 들어 Stm32CubeIde 에서 링커 옵션에 'Discard unused sections' 체크박스를 비활성화하시기 바랍니다.

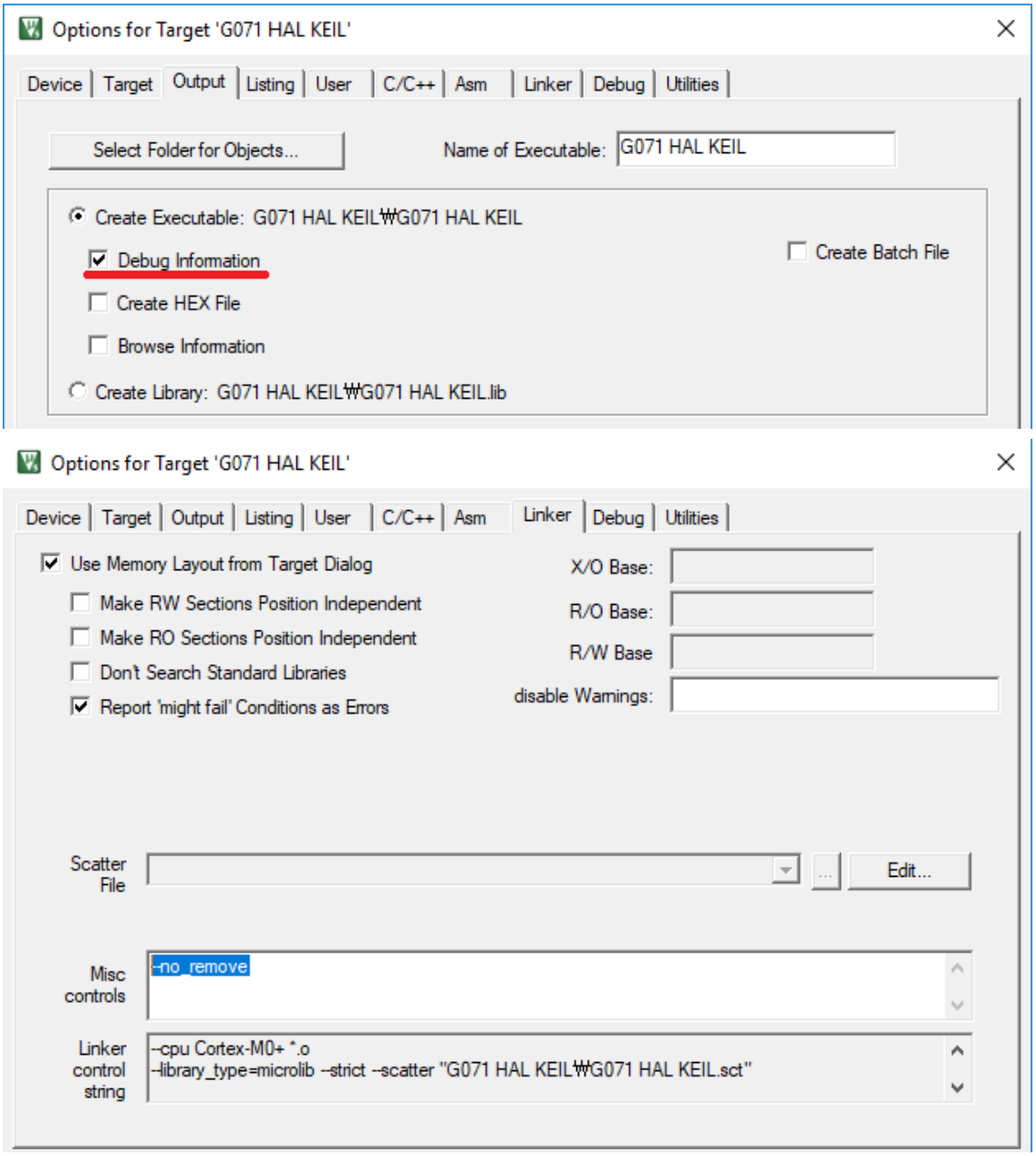
다른 개발 환경이라면 적절히 해당하는 세팅이 필요합니다.

예제 : Stm32CubeIDE

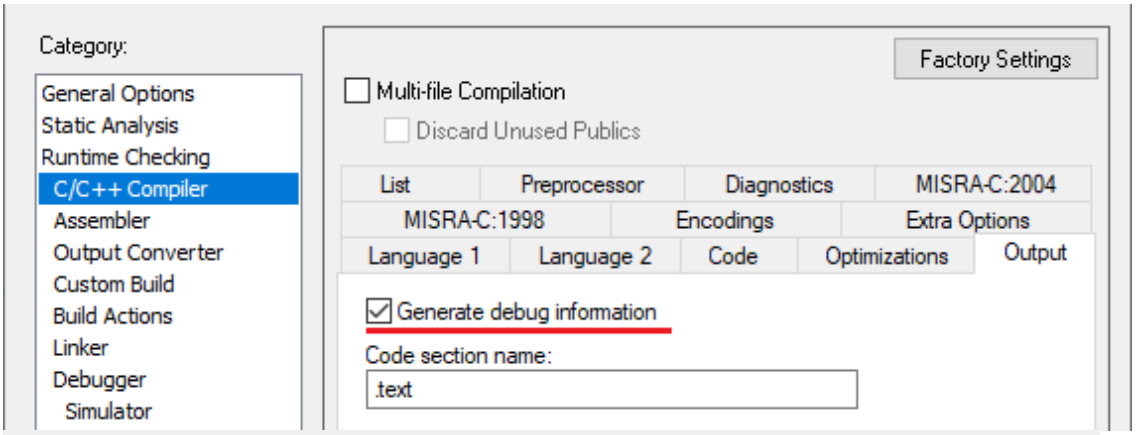


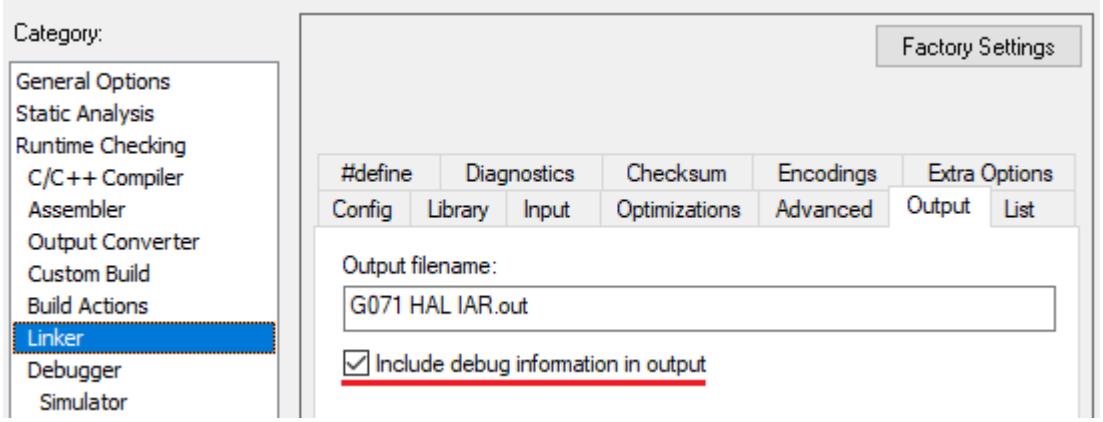


예제 : KEIL uVision



예제 : IAR Embedded Workbench





7.2.2 STM32 하드웨어 설정

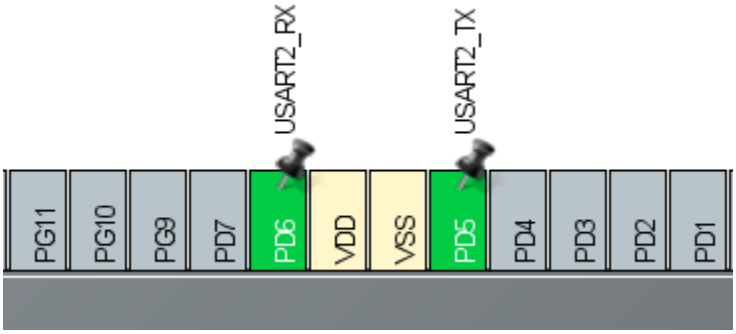
STEP 1 : USART 채널 및 핀 선정

easyDSP 는 MCU 와의 통신 및 부트로더(플래시 및 램부팅 관련 동작)를 사용하기 위해 USART 통신을 사용합니다.
MCU 종류에 적절히 USART 채널 하나를 easyDSP 통신용으로 할당해야 합니다.
ST 의 어플리케이션 노트(AN2606 : STM32 microcontroller system memory boot mode)을 참조하셔서 부트로더가 지원되는 USART 채널 및 핀을 선정하여 주세요. UART 채널은 사용할 수 없습니다.
(STM32WL3x 경우 2025 년 4 월 기준으로 AN2606 에 해당 정보가 없으므로, QFN48 패키지 경우 USART1 Rx = PA15, USART1 Tx = PA1, QFN32 패키지 경우 USART1 Rx = PB14, USART1 Tx = PA1 을 사용해주세요)
예를 들어, STM32F413x 의 경우 하기 테이블과 같이 총 3 개의 USART 를 부트로더용으로 제공하고 있습니다. 이중 하나를 적절히 선택하시면 됩니다.

Table 67. STM32F413xx/423xx configuration in system memory boot mode (continued)

Bootloader	Feature/Peripheral	State	Comment
USART1 bootloader	USART1	Enabled	Once initialized the USART1 configuration is: 8-bit, even parity and 1 Stop bit
	USART1_RX pin	Input	PA10 pin: USART1 in reception mode
	USART1_TX pin	Output	PA9 pin: USART1 in transmission mode
USART2 bootloader	USART2	Enabled	Once initialized the USART2 configuration is: 8-bit, even parity and 1 Stop bit
	USART2_RX pin	Input	PD6 pin: USART2 in reception mode
	USART2_TX pin	Output	PD5 pin: USART2 in transmission mode
USART3 bootloader	USART3	Enabled	Once initialized the USART3 configuration is: 8-bit, even parity and 1 Stop bit
	USART3_RX pin	Input	PB11 pin: USART3 in reception mode
	USART3_TX pin	Output	PB10 pin: USART3 in transmission mode

만약 USART2 를 선택했다면 반드시 PD5, PD6 번 핀을 사용해야 합니다. STM32CubeMX 에서도 해당 핀을 USART 로 설정해야 합니다 .



주의 사항 : 하기 조합 사용시 발생되는 제약점 주의 바랍니다. 가능하다면 다른 조합으로 사용하면 좋습니다.

MCU	USART	핀	제한 사항
STM32F03xx4/6	USART1	PA14 PA15	부트로더에 진입한 이후 SWD 통신 불가 . PA14(SWCLK)이 부트로더에서 사용되기 때문.
STM32F030xC STM32F05xxx STM32F030x8 STM32F04xxx STM32F070x6 STM32F070xB STM32F071xx STM32F072xx STM32F09xxx	USART2	PA14 PA15	부트로더에 진입한 이후 SWD 통신 불가 . PA14(SWCLK)이 부트로더에서 사용되기 때문.

주의 사항 : MCU 내장 부트로더도 일종의 프로그램으로 버그가 많이 리포팅되고 있습니다. 특히 오래된 부트로더가 탑재된 일부 MCU 가 그렇습니다.

이로 인해 하기 조합에서 부트로더 진입이 불가능하므로 사용을 피하여 주십시요.

상세한 사항은 AN2606 을 참조하세요.

리포팅되지 않은 버그로 인해, 하기 조합이 아니더라도 부트로더에 진입이 불가능할 수 있으니 유의 부탁드립니다.

MCU	BL ID	USART
STM32F105xx/107xx	V2.0 (0x20)	USART1,USART2
STM32F412xx	V9.0 (0x90)	USART3
STM32G05xxx/061xx	V5.0 (0x50)	USART2
STM32H74xxx STM32H75xxx	V13.2 (0xD2)	USART2
STM32L552xx STM32L562xx	V13.0 (0xD0)	USART3
STM32L47xxx/48xxx	V9.2 (0x92)	USART2
STM32L496xx/4A6xx	V9.3 (0x93)	USART2, USART3
STM32L4P5xx/Q5xx	V9.0 (0x90)	USART2, USART3
STM32L4Rxx/4Sxx	V9.2 (0x92)	USART2, USART3
STM32L4RxG/4SxG	V9.2 (0x92)	all USARTx 결국 사용 불가

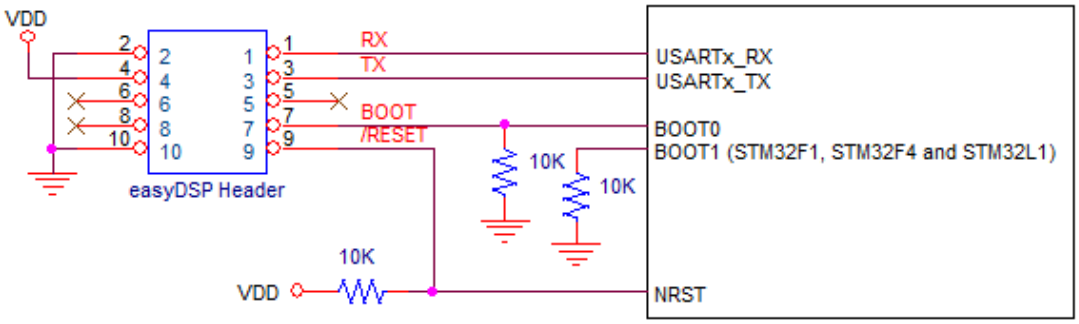
STEP 2 : easyDSP 연결

상기 STEP 1 에서 선정된 USART 채널 및 해당 핀에 easyDSP 를 하기와 같이 연결합니다 .

easyDSP 헤더 4 번핀에는 MCU 의 VDD 를 연결하여 주십시요.

STM32F1, STM32F4 및 STM32L1 경우 BOOT1 핀을 풀다운해주세요.

RX, TX 신호는 easyDSP 포드 내에서 100k 오옴으로 풀업되어 있습니다.



주의 사항)

STM32WB0, STM32WL33xx : BOOT0 핀은 PA10 입니다.

STM32H74xxx/75xxx : PB15 단자를 풀다운하지 마시기 바랍니다.

STM32G03xx/04xxx : 오래된 부트로더 버전일 경우 (v5.1, v5.2), PA3 단자를 풀다운하지 마시기 바랍니다.

STM32C011xx : WLCSP12, SO8N, TSSOP20, UFQFN20 패키지에서는 USART1 PA9/PA10 핀이 PA11/PA12 에 Remapping 되어 있습니다.

STM32C031xx : TSSOP20, UFQFN28 패키지에서는 USART1 PA9/PA10 핀이 PA11/PA12.에 Remapping 되어 있습니다.

STEP 3 : MCU option byte 설정

필요시 easyDSP 가 MCU 를 부트로더에 진입시키고, 램 부팅 또는 플래시 프로그래밍의 동작을 수행하기 위해서는, MCU 의 옵션바이트가 적절히 미리 세팅되어 있어야 합니다 .

easyDSP 는 옵션바이트를 설정할 수 없으므로 다른 프로그래밍 툴을 사용하셔서 하기 설정을 준수하여 주십시요 .

아래 설명은 STM32CubeProgrammer 기준이며 MCU 에 따라 일부 옵션이 생략될 수 있습니다.

easyDSP 가 억세스할 메모리 영역에 하기 각종 보호 장치가 비활성화되어 있어야 합니다.

즉, 하기 각종 보호 장치를 사용하지 않습니다.

- WRP (Write Protect)
- RDP (Readout Protect)
- PCROP (Proprietary code read-out protection)
- Securable memory

easyDSP 가 램부팅 또는 플래시를 라이팅하기 위해 MCU 를 부트로더에 진입 (system memory 부팅)시킬 수 있는 설정이 되어 있어야 합니다.

만약 easyDSP 가 변수 통신만을 위해 사용된다면 하기 설정은 필요 없습니다 .

- BOOT_LOCK 은 사용되지 말아야 합니다 : BOOT_LOCK = unchecked

BOOT_LOCK	☐	used to force boot from user area Unchecked : Boot based on the pad/option bit configuration Checked : Boot forced from Main Flash memory

BOOT_LOCK	☐	Unchecked : CPU1 CM4 Boot lock disabled Checked : CPU1 CM4 Boot lock enabled
C2BOOT_LOCK	☐	Unchecked : CPU2 CM0+ Boot lock disabled Checked : CPU2 CM0+ Boot lock enabled

- Product State 기능이 있는 MCU 라면, 플래시 프로그래밍이 가능하기 위해서는 Product State 는 Open 이어야 합니다.

PRODUCT_STATE	ED	Life state code. ED : Open 17 : Provisioning 2E : Provisioned 72 : Closed

- NRST 핀이 easyDSP 로부터 리셋 신호를 입력 받을 수 있어야 합니다 : NSRT_MODE = 1 내지 3

NRST_MODE	3	0 : Reserved 1 : Reset Input only: a low level on the NRST pin generates system reset, internal RESET not propagated to the NSRT pin 2 : GPIO: standard GPIO pad functionality, only internal RESET possible 3 : Bidirectional reset: NRST pin configured in reset input/output mode (legacy mode)

- MCU 의 부트모드는 BOOT0 핀에 의해 결정되어야 합니다 (BOOT0 핀 low 면 플래시부팅, BOOT0 핀 high 면 시스템 메모리(부트로더) 부팅) :

nBOOL_SEL = unchecked, BOOT_SEL = checked, nBOOT1 = checked, nSWBOOT0 = checked

nSWBOOT0	☑	Software BOOT0 Unchecked : BOOT0 taken from the option bit nBOOT0 Checked : BOOT0 taken from PB8/BOOT0 pin
nBOOT1	☑	Unchecked : Boot from Flash if BOOT0 = 0, otherwise Embedded SRAM1 Checked : Boot from Flash if BOOT0 = 0, otherwise system memory
BOOT_SEL	☑	Unchecked : BOOT0 signal is defined by nBOOT0 option bit Checked : BOOT0 signal is defined by BOOT0 pin value
nBOOT_SEL	☐	Unchecked : BOOT0 signal is defined by BOOT0 pin value (legacy mode) Checked : BOOT0 signal is defined by nBOOT0 option bit

- MCU 가 리셋시 BOOT0 신호에 따라 설정된 주소로 부팅할 경우, 각각의 주소가 플래시, 시스템 메모리 (부트로더)로 설정되어야 합니다 .

예를 들어 STM32H7A3 의 경우,

Name	Value			
BOOT_CM7_ADD0	Value	0x800	Address 0x8000000	Define the boot address for Cortex-M7 when BOOT0=0
BOOT_CM7_ADD1	Value	0x1ff0	Address 0x1ff00000	Define the boot address for Cortex-M7 when BOOT0=1

STM32F767 의 경우라면

Name	Value			
BOOT_ADD0	Value	0x80	Address 0x00200000	Define the boot address when BOOT0=0
BOOT_ADD1	Value	0x40	Address 0x00100000	Define the boot address when BOOT0=1

- STM32H7 dual core MCU 의 경우 모든 코어가 부트 활성화되어야 합니다.

BCM4	☑	Unchecked : CM4 boot disabled Checked : CM4 boot enabled
BCM7	☑	Unchecked : CM7 boot disabled Checked : CM7 boot enabled

STM32WL dual core MCU 의 경우

C2OPT	☑	Unchecked : SBRV will address SRAM1 or SRAM2, from start address 0x2000 0000 + SBRV. Checked : SBRV will address Flash memory, from start address 0x0800 0000 + SBRV.

7.2.3 STM32 듀얼 코어

두개의 코어가 존재할 경우, 앞서 설명된 소프트웨어/하드웨어 설정 방식에 덧붙여, 추가적인 주의사항이 필요합니다.

대상 MCU

STM32H745x, STM32H747x, STM32H755x, STM32H757x (CPU1 = Arm Cortex-M7, CPU2 = Arm Cortex-M4)
STM32WL55xx, STM32WL54xx (CPU1 = Arm Cortex-M4, CPU2 = Arm Cortex-M0+)

공통 사항

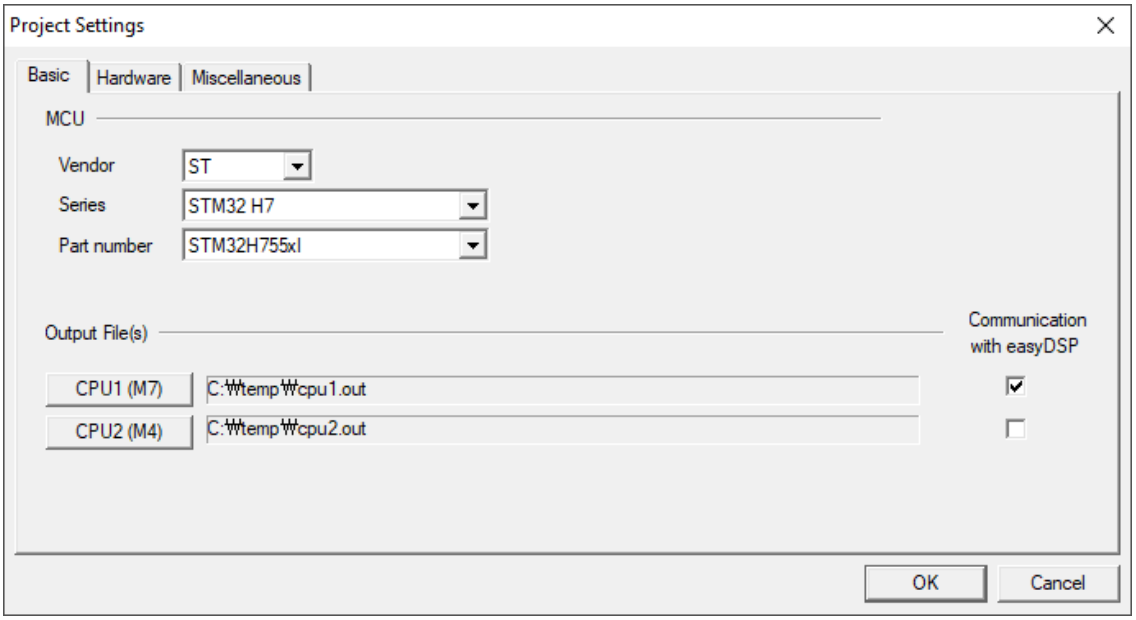
easyDSP 측면에서 코어는 총 4 가지 종류로 구분됩니다.
노란색 코어 : easyDSP Pod 가 연결되고 easyDSP 모니터링 수행되는 코어
주황색 코어 : easyDSP Pod 가 연결되지는 않지만 easyDSP 모니터링이 수행되는 코어
파란색 코어 : easyDSP 모니터링이 수행되지 않는 코어
회색 코어 : 동작하지 않는 코어입니다.

	Connected to easyDSP pod	Communicated with easyDSP	Running core
core	**Yes**	**Yes**	**Yes**
core	No	**Yes**	**Yes**
core	No	No	**Yes**
core	No	No	No

STM32 시리즈는 최대 2 개의 코어가 제공되므로, 사용자 시스템에 맞춰 해당 코어를 선정하세요.
파란색 코어와 회색 코어는 easyDSP 와 관련 없으므로 easyDSP 관련 작업을 수행하지 않습니다.
따라서 노란색 코어, 주황색 코어를 선정해야 합니다.

모든 노란색 코어에 대해서는 easyDSP 프로젝트가 생성되어야 합니다. easyDSP 프로젝트에서 실제 사용자 프로그램에서 사용하는 모든 코어의 출력 파일을 지정할 수 있습니다. 2 개의 코어를 모두 사용한다면 2 개의 출력 파일을 지정할 수 있습니다. easyDSP 는 지정된 출력파일 기준으로 플래시 프로그래밍을 수행합니다.
그리고 easyDSP 가 모니터링하는 코어에 대해서, 즉, easyDSP 를 사용하여 각종 변수 모니터링을 수행할 코어에 대해서, 체크 박스를 선정합니다.

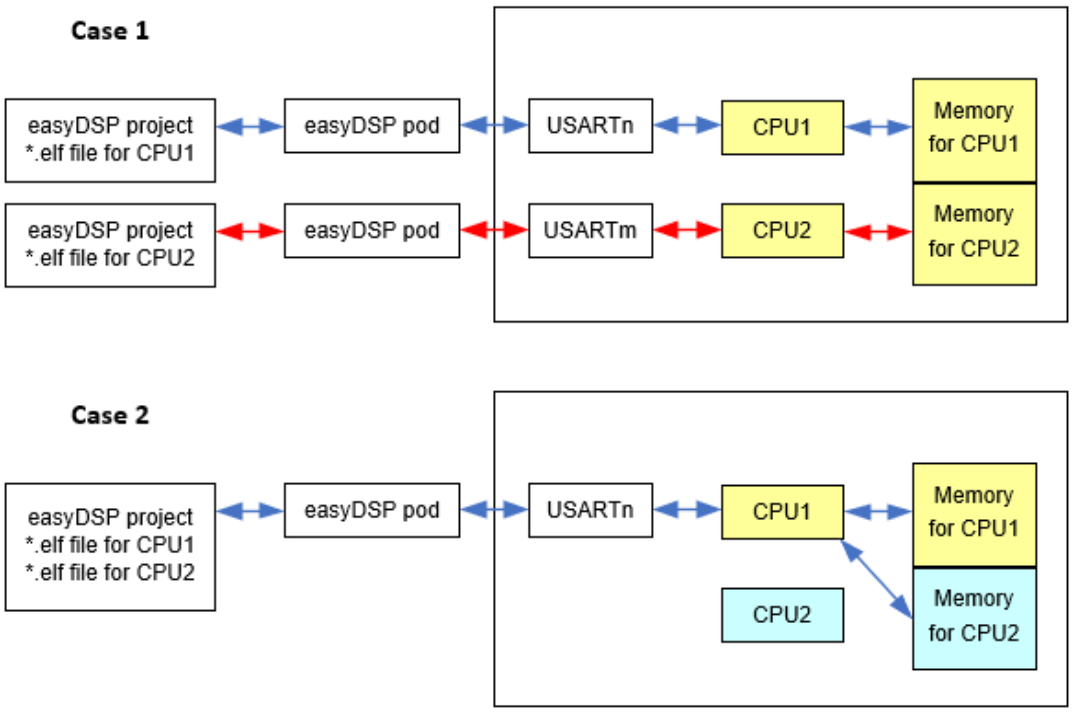
하기 경우는 2 개의 코어가 동작하지만 easyDSP 는 CPU1 에 대해서만 모니터링을 수행하는 경우입니다



easyDSP 가 2 개의 코어와 통신할 때 (즉, 체크 박스가 모두 체크), easyDSP 내에서 코어간 변수 이름을 구분하기 위해, CPU1 프로그램 변수에는 "1:"을, CPU2 프로그램 변수에는 "2:"를 기존 변수 이름 앞에 덧붙입니다. 즉, CPU1 프로그램 변수 이름이 "var1" 일 경우, easyDSP 에서 "1:var1" 으로 표기되며, CPU2 프로그램 변수 이름이 "var2" 일 경우, easyDSP 에서 "2:var2" 으로 표기됩니다.

STM32WL dual core

하기와 같이 총 2 가지 결선 구성이 가능합니다. 그림내 화살표는 easyDSP 모니터링을 위한 데이타 전송 경로를 나타냅니다.



Case 1 :

각각의 CPU 에 easyDSP Pod 를 연결하는 방식으로 각각의 CPU 에 대해 앞서 설명된 설정 방식을 수행합니다. 하나의 CPU 만 사용할 수도 있습니다.
아래 출력 파일 지정 선택사항을 수행하면 각 easyDSP 프로젝트에서 CPU1, CPU2 모두의 플래시 프로그래밍이 가능합니다.
선택 사항을 하지 않으면 각 easyDSP 프로젝트에서 각 CPU 의 플래시 프로그래밍만 가능합니다.

설정 항목	CPU1	CPU2
easyDSP 프로젝트	CPU1 출력 파일만 지정 CPU2 출력 파일 지정은 선택 사항 CPU1 체크박스만 선택	CPU2 출력 파일만 지정 CPU1 출력 파일 지정은 선택 사항 CPU2 체크박스만 선택
main.c	easyDSP_init(USARTn) 호출	easyDSP_init(USARTm) 호출
stm32h7xx_it.c	USARTx_IRQHandler()에서 ez_USARTx_IRQHandler() 호출	USARTx_IRQHandler()에서 ez_USARTx_IRQHandler() 호출

Case 2 :

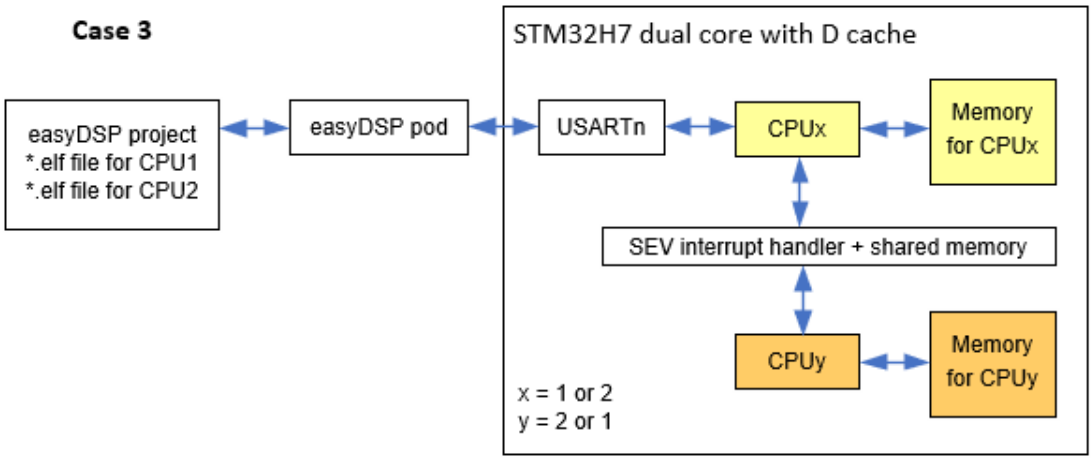
CPU1 에만 easyDSP Pod 를 연결하고, CPU1 을 통해 (CPU2 는 불가) 전체 메모리 영역을 억세스하는 방식입니다. 따라서 마치 싱글 코어 경우와 마찬가지로, 앞서 설명된 설정 방식 을 CPU1 프로젝트에 반영합니다. CPU2 에서는 easyDSP 관련 아무 작업도 하지 않습니다.

설정항목	CPU1

easyDSP 프로젝트	CPU1, CPU2 출력 파일 모두 지정 CPU1, CPU2 체크 박스 모두 선택
main.c	easyDSP_init(USARTn) 호출
stm32h7xx_it.c	USARTx_IRQHandler()에서 ez_USARTx_IRQHandler() 호출

STM32H7 dual core

데이타 캐시 사용 여부에 따라 (Stm32CubeMx > System Core > CORTEX_M7 > Parameter Settings > Cortex Interface Settings > CPU DCache) 하기 그림과 같이 Case 3 를 포함하여 총 3 가지 결선 구성이 가능합니다.
그림내 화살표는 easyDSP 모니터링을 위한 데이타 전송 경로를 나타냅니다.



Case 1 :

각각의 CPU 에 각각의 easyDSP Pod 를 연결하는 방식으로 데이타 캐시 사용여부와 관련없이 동작하는 장점이 있습니다. 각각의 CPU 에 대해 앞서 설명된 설정 방식을 수행합니다.
하나의 CPU 만 사용할 수도 있습니다.
아래 출력 파일 지정 선택사항을 수행하면 각 easyDSP 프로젝트에서 CPU1, CPU2 모두의 플래시 프로그래밍이 가능합니다.
선택 사항을 하지 않으면 각 easyDSP 프로젝트에서 각 CPU 의 플래시 프로그래밍만 가능합니다.

설정 항목	CPU1	CPU2
easyDSP 프로젝트	CPU1 출력 파일만 지정 CPU2 출력 파일 지정은 선택 사항 CPU1 체크박스만 선택	CPU2 출력 파일만 지정 CPU1 출력 파일 지정은 선택 사항 CPU2 체크박스만 선택
easyStm32LL.h	EZ_DUAL_CORE = 1 EASYDSP_IS_CONNECTED_TO_THIS_CORE = 1	EZ_DUAL_CORE = 1 EASYDSP_IS_CONNECTED_TO_THIS_CORE = 1

	EZ_USE_SEV_INT = 0	EZ_USE_SEV_INT = 0
main.c	easyDSP_init(USARTn) 호출	easyDSP_init(USARTm) 호출
stm32h7xx_it.c	USARTx_IRQHandler()에서 ez_USARTx_IRQHandler() 호출	USARTx_IRQHandler()에서 ez_USARTx_IRQHandler() 호출

Case 2 :

데이타 캐시가 사용되지 않을 경우, easyDSP 는 CPU1 를 통해 (CPU2 는 불가) 전체 메모리 영역을 억세스할 수 있습니다.
따라서 마치 싱글 코어 경우와 마찬가지로, 앞서 설명된 설정 방식 을 CPU1 프로젝트에 반영합니다. CPU2 에서는 easyDSP 관련 아무 작업도 하지 않습니다.

설정항목	CPU1
easyDSP 프로젝트	CPU1, CPU2 출력 파일 모두 지정 CPU1, CPU2 체크 박스 모두 선택
easyStm32LL.h	EZ_DUAL_CORE = 1 EASYDSP_IS_CONNECTED_TO_THIS_CORE = **1** EZ_USE_SEV_INT = 0
main.c	easyDSP_init(USARTn) 호출
stm32h7xx_it.c	USARTx_IRQHandler()에서 ez_USARTx_IRQHandler() 호출

Case 3 :

데이타 캐시가 사용될 경우 Case 2 구성은 캐시 일관성 문제를 발생시킵니다. 이를 방지하기 위해, 즉 해당 CPU 가 해당 CPU 변수가 위치한 메모리를 억세스하기 위해 CPU 간 통신으로 SEV 인터럽트 및 공유 메모리를 사용하고 있습니다.
easyDSP pod 는 CPU1, CPU2 어느 쪽이든 연결 가능하오니 사용자 어플리케이션에 맞게 선택 바랍니다.

STM32CubeMx 에서 System Core > NVIC1 및 NVIC2 에서 SEV 인터럽트를 최하위 순위로 활성화합니다.

NVIC1 Interrupt Table	Enabled	Preemption Priority
Non maskable interrupt	☑	0
Hard fault interrupt	☑	0
Memory management fault	☑	0
Pre-fetch fault, memory access fault	☑	0
Undefined instruction or illegal state	☑	0
System service call via SWI instruction	☑	0
Debug monitor	☑	0
Pendable request for system service	☑	0
Time base: System tick timer	☑	0
USART3 global interrupt	☑	15
CM4 send event interrupt for CM7	☑	15

NVIC2 Interrupt Table	Enabled	Preemption Priority
Non maskable interrupt	☑	0
Hard fault interrupt	☑	0
Memory management fault	☑	0
Pre-fetch fault, memory access fault	☑	0
Undefined instruction or illegal state	☑	0
System service call via SWI instruction	☑	0
Debug monitor	☑	0
Pendable request for system service	☑	0
Time base: System tick timer	☑	0
USART1 global interrupt	☑	15
CM7 send event interrupt for CM4	☑	15

공유 메모리 위치는 어디든 설정 가능하지만 SRAM4 를 추천드립니다. 주의사항은 하기와 같습니다.

1. 해당 영역 (시작번지부터 32 바이트)은 CPU1/CPU2 모두에서 다른 용도로 사용되면 안되므로 CPU1/CPU2 링커 스크립트 파일 사용에 주의 바랍니다.
2. 시작 번지는 32B 에 얼라인되어야 합니다 (예를 들어 0x38000000 또는 0x38000020)
3. 해당 영역은 non cacheable 로 지정되어야 하며 하기와 같은 Stm32CubeMx 의 System Core > CORTEX_M7 에서 MPU 설정이 필요합니다.

MPU region 우선 순위 (Region 번호가 높을 수록 우선 순위)에 의해 설정된 값이 무효화되지 않도록 주의 바랍니다.

Cortex Memory Protection Unit Region 0 Settings

MPU Region	Enabled
MPU Region Base Address	0x38000000
MPU Region Size	32B
MPU TEX field level	level 1
MPU Access Permission	ALL ACCESS PERMITTED
MPU Instruction Access	DISABLE
MPU Shareability Permission	ENABLE
MPU Cacheable Permission	DISABLE
MPU Bufferable Permission	DISABLE

그리고 easyDSP 가 연결된 CPU, 연결되지 않은 CPU 프로젝트 모두에 easyDSP 소스파일을 추가하며 하기 테이블처럼 설정합니다.

설정항목	easyDSP pod 가 연결된 CPU	easyDSP pod 가 연결되지 않은 CPU

easyDSP 프로젝트	CPU1, CPU2 출력 파일 모두 지정 CPU1, CPU2 체크 박스 모두 선택	프로젝트 생성하지 않음
easyStm32LL.h	EZ_DUAL_CORE = 1 EASYDSP_IS_CONNECTED_TO_THIS_CORE = **1** EZ_USE_SEV_INT = 1 EZ_SHARED_MEM_ADDRESS = 사용자 설정값	EZ_DUAL_CORE = 1 EASYDSP_IS_CONNECTED_TO_THIS_CORE = **0** EZ_USE_SEV_INT = 1 EZ_SHARED_MEM_ADDRESS = 사용자 설정값
main.c	easyDSP_init(USARTn) 호출	easyDSP_init(**0**) 호출
stm32h7xx_it.c	USARTx_IRQHandler()에서 ez_USARTx_IRQHandler() 호출 CMx_SEV_IRQHandler()에서 ez_SEV_IRQHandler() 호출	CMx_SEV_IRQHandler()에서 ez_SEV_IRQHandler() 호출

7.2.4 STM32 램부팅 관련 설정

만약 램부팅을 사용하지 않는다면 본 설정은 필요하지 않습니다 .

램 사이즈가 매우 작은 MCU 는 아예 램 부팅이 불가능합니다만. 일부 MCU 의 경우 충분한 램 사이즈를 활용하여 flash 를 사용하지 않고 램부팅이 가능합니다 .

easyDSP 는 램부팅을 위해 MCU 의 부트로더를 사용합니다. 따라서 디버거를 사용한 램부팅과의 차이점 그리고 제한 사항은 모두 부트로더에서 기인합니다.

하기 참조하시기 바랍니다.

<table>
<tr><th>단계</th><th>예제</th></tr>
<tr><td>1. 제한 사항</td><td>1. 특정 MCU 경우 램부팅이 원칙적으로 지원되지 않습니다. 하기 MCU 입니다.
STM32F04xxx
STM32F070x6
STM32L01xxx/02xxx
STM32L031xx/041xx

2. 특정 MCU 에 내장된 부트로더의 버전이 오래된 경우, 램 부팅이 불가능할 수도 있습니다. 하기 표 조합 참조하세요. 최신 부트로더가 탑재된 경우 이러한 제한 사항이 없습니다. 상세 사항은 최신 버전의 AN2606(STM32 microcontroller system memory boot mode)를 참조하세요.
<table>
<tr><th>MCU</th><th>Bootloader version</th></tr>
<tr><td>STM32H74xxx</td><td>V13.2 (0xD2)</td></tr>
</table>
</td></tr>
</table>

STM32H75xxx	
STM32L552xx STM32L562xx	V13.0 (0xD0)
STM32L47xxx STM32L48xxx	V10.1 (0xA1) V9.0 (0x90)
STM32F100xx STM32F101xx STM32F102xx STM32F103xx (단, STM32F101xF, STM32F101xG, STM32F103xF, STM32F103xG 제외)	V2.0 (0x20)

3. dual core 제품 (H745, H747, H755, H757, WL5x)의 경우 램부팅을 지원하지 않습니다.

2. 링커 스크립트 파일의 램 영역 수정

부트로더도 일종의 프로그램이기에 사용하는 램 영역이 있습니다. 따라서 사용자 프로그램 램 영역과 충돌되면 안됩니다 . 또한 부트로더 진입시 모든 램 영역을 억세스할 수 있는 것도 아닙니다.

따라서 램 부팅에 사용될 수 있는 램 영역은 제한되며, MCU 별로 사용 가능한 램 영역에 대해서는 최신 버전의 AN2606(STM32 microcontroller system memory boot mode)를 참조하세요 .

충돌 방지를 위해 링커 스크립트 파일 (예 : STM32CubeIDE 에서 STM32F413 사용시 파일 이름은 STM32F413ZHTX_RAM.ld) 내의 램 메모리

Table 145. Bootloader device-dependent parameters (continued)

STM32 Series	Device	PID	BL ID	RAM	System memory
F4	STM32F40xxx/41xxx	0x413	0x31	0x20002000 - 0x2001FFFF	0x1FFF0000 - 0x1FFF77FF
			0x90	0x20003000 - 0x2001FFFF	
	STM32F42xxx/43xxx	0x419	0x70	0x20003000 - 0x2002FFFF	
			0x91		
	STM32F401xB(C)	0x423	0xD1	0x20003000 - 0x2000FFFF	
	STM32F401xD(E)	0x433	0xD1	0x20003000 - 0x20017FFF	
	STM32F410xx	0x458	0xB1	0x20003000 - 0x20007FFF	
	STM32F411xx	0x431	0xD0	0x20003000 - 0x2001FFFF	
	STM32F412xx	0x441	0x90	0x20003000 - 0x2003FFFF	
	STM32F446xx	0x421	0x90	0x20003000 - 0x2001FFFF	
	STM32F469xx/479xx	0x434	0x90	0x20003000 - 0x2005FFFF	
	STM32F413xx/423xx	0x463	0x90	0x20003000 - 0x2004FFFF	

따라서 STM32F413ZHTX_RAM.ld 파일내

```
/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 320K
  FLASH  (rx)     : ORIGIN = 0x8000000,    LENGTH = 1536K
}
```

부분을 하기와 같이 수정합니다. 12K 바이트가 사용되므로 사용자 프로그램용 램 영역이

영역을 수정해야 합니다.

0x20003000 번지 이후로 변경되었습니다 .

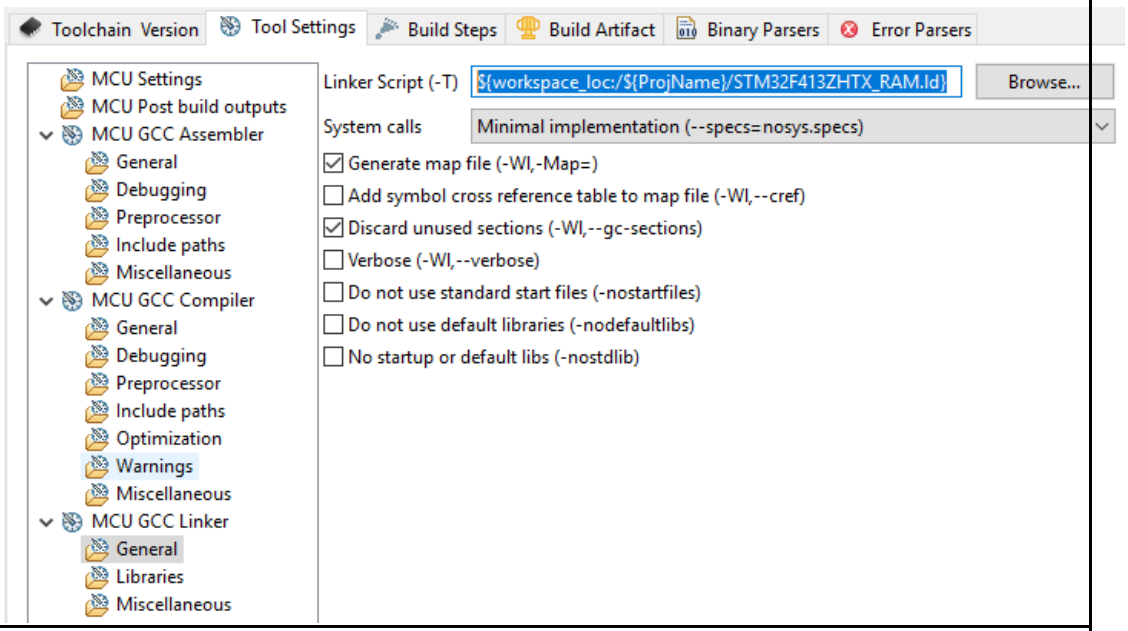
```
/* Memories definition for RAM booting*/
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20003000,   LENGTH = 308K
  FLASH    (rx)   : ORIGIN = 0x8000000,   LENGTH = 1536K
}
```

3. 벡터 테이블을 램 영역 첫 번지에 위치하도록 링커 스크립트 파일 수정

easyDSP 는 램부팅 사용시 ISR 벡터 테이블이 램 영역 첫 번지에 있다고 가정합니다. 따라서 이에 맞게 필요시 링커 스크립트 파일을 수정해야 합니다. Stm32CubeIde 의 경우, 디폴트로 ISR 벡터 테이블을 SECTIONS 의 맨 앞에 위치시키므로서 이러한 조건을 이미 충족시키므로 부가적인 작업을 필요가 없습니다. 다른 IDE 를 사용하시거나 디폴트 링커 스크립트를 사용하지 않는 경우에 주의 부탁드립니다.

```
/* Sections */
SECTIONS
{
  /* The startup code into "RAM" Ram type memory */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >RAM
```

4. 수정된 링커 스크립트 파일 을 컴파일러에 등록



5. 벡터테이블 주소 변경 system_stm32(MCU 이름).c 파일을 수정합니다.

STM32F413 의 경우라면

system_stm32f4xx.c 파일 수정 전 : 플래시 부팅을 위해, VECT_TAB_SRAM 이 정의되어 있지 않으며 VECT_TAB_OFFSET 은 0 으로 정의되어 있습니다 .

```
/* #define VECT_TAB_SRAM */
#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                                   This value must be a multiple of 0x200. */
```

<table>
<tr>
<td>system_stm32f4xx.c
파일을 수정합니다.</td>
<td>system_stm32f4xx.c 파일 수정 후 : 램부팅을 위해, VECT_TAB_SRAM 을 정의하고 VECT_TAB_OFFSET 은 0x3000 으로 정의합니다 .
MCU 에 따라 USER_VECT_TAB_ADDRESS 도 정의할 필요가 있습니다 . (ex, STM32L5, STM32U3)
여기서 0x3000 은 앞 서 링크 스크립트 파일에서 변경된 램 영역(0x20003000 부터)과 관련됩니다. 아래 예제처럼 VECT_TAB_SRAM 의 정의 유무에 따라 VECT_TAB_OFFSET 값을 0 또는 0x3000 으로 변경하면 램부팅/플래시부팅간에 좀 더 손쉬운 전환이 가능합니다. 즉, VECT_TAB_SRAM 유무가 곧 램부팅/플래시부팅을 결정하게 됩니다 .

<pre>
#ifdef VECT_TAB_SRAM
#define VECT_TAB_OFFSET 0x3000
#else
#define VECT_TAB_OFFSET 0x00 /*!< Vector Table base offset field.
                                   This value must be a multiple of 0x200. */
#endif
</pre>

MCU 종류에 따라 추가로 고려해야 할 사항이 있을 수 있습니다. 해당 MCU 의 system_stm32yyxx.c 파일의 해당 부분을 잘 참조하십시요.</td>
</tr>
<tr>
<td>6. 기타 설정</td>
<td>MCU 에 내장된 부트로더의 버전에 따라 특정 MCU, 특정 부트로더 조합에서 추가 고려 사항을 하기에 표시합니다 .

경우 1 : STM32H74x/H75x 에 내장된 BL 버전이 V9.0 일 경우
하기와 같이 STM32H743ZITX_RAM.ld 에 설정되어 있는 스택 포인터를
<pre>_estack = ORIGIN(RAM_D1) + LENGTH(RAM_D1); /* end of "RAM_D1" Ram type memory */</pre>

하기와 같이 -16 을 추가하여 스택 포인터가 램 끝위치보다 16 만큼 작게 설정합니다.
<pre>_estack = ORIGIN(RAM_D1) + LENGTH(RAM_D1) - 16; /* Application stack pointer must be lower than (RAM end @ - 16 bytes) */</pre>

경우 2 : STM32WB55 경우, *_RAM.ld 파일 끝 부분에 하기 3 줄을 삽입해야 합니다 .
MAPPING_TABLE (NOLOAD) : { *(MAPPING_TABLE) } >RAM_SHARED
MB_MEM1 (NOLOAD) : { *(MB_MEM1) } >RAM_SHARED
MB_MEM2 (NOLOAD) : { _sMB_MEM2 = . ; *(MB_MEM2) ; _eMB_MEM2 = . ; }

경우 3 : STM32WBA, STM32U545 경우 NEW
하기와 같이 STM32WBA52CGUX_RAM.ld (또는 STM32U545RETXQ_RAM.ld)에 설정되어 있는 스택 포인터에
_estack = ORIGIN(RAM) + LENGTH(RAM);
하기와 같이 -16 을 추가하여 스택 포인터가 램 끝위치보다 16 만큼 작게 설정합니다.
_estack = ORIGIN(RAM) + LENGTH(RAM) - 16;</td>
</tr>
</table>

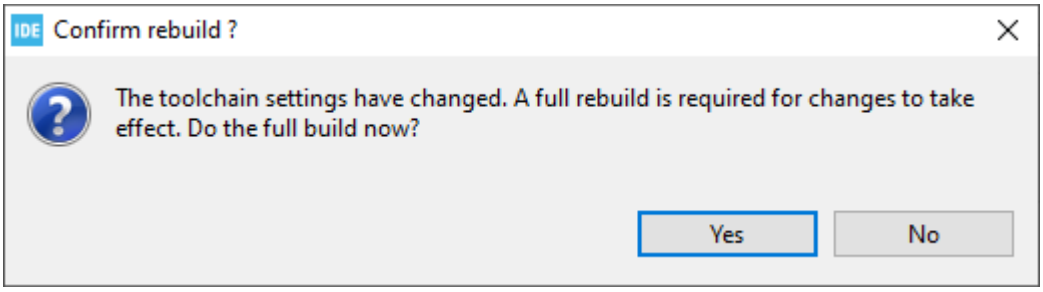
7.2.5 STM32 주의사항

MCU 가 부트로더 진입시 일부 IO 를 출력핀으로 설정하는 것을 고려한 보드 설계

MCU 는 여러 경우 부트로더로 진입하게 됩니다. easyDSP 사용시 램부팅 또는 플래시 작업을 하기 위한 경우입니다. 또한 일부 MCU 의 경우, MCU 가 프로그램되어 있지 않으면 전원 투입시 부트로더로 진입하게 됩니다. ST 의 어플리케이션 노트(AN2606 : STM32 microcontroller system memory boot mode) 참조하시면 MCU 가 부트로더 진입시 일부 통신 IO 를 출력핀으로 설정함을 알 수 있습니다. 따라서 이 IO 핀들이 출력핀이 되더라도 보드의 다른 부분에 데미지를 주지 않도록 설계해야 합니다. 예를 들어 해당 핀이 GND 나 VDD 에 직결된 경우 데미지가 발생할 수 있습니다.

STM32CubeIDE 사용시 Full Rebuild 주의

컴파일 설정에 주요 변경시 Full rebuild 가 요구되는데 (아래 그림 참조), 이 때 컴파일러 출력 폴더의 내용이 전부 지워지게 됩니다. 만약 easyDSP 프로젝트를 컴파일러 출력 폴더에 지정하셨다면 easyDSP 프로젝트 관련 파일도 전부 지워지게 됩니다.



통신 보드레이트 관련

USART 인터럽트 통신에 필요한 시간 리소스가 충분하지 않은 경우에는 높은 통신 bps 가 통신 불량을 유발할 수 있습니다. 이 경우 OverRun Error 가 발생됩니다.

MCU 이름에 뱅크 모드 포함

MCU 이름에 single/dual bank 설정이 포함된 경우가 있는데, 이는 single 또는 dual bank 를 구분해야 하는 경우에만 포함됩니다. 즉, 항상 single bank 이거나 항상 dual bank 이거나 또는 single 및 dual bank 를 구분할 필요가 없는 경우에는 표시되지 않습니다.

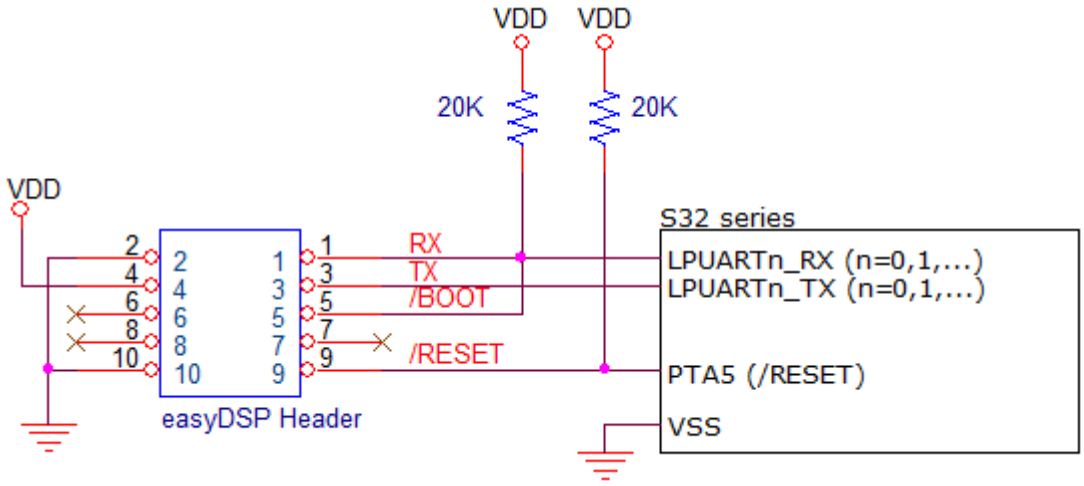
7.3 S32

7.3.1 S32K1 + SDK 사용

본 페이지는 사용자가 S32 Configuration Tools 및 S32K1 SDK API 를 사용함을 전제하고 있습니다.

STEP 1 : 하드웨어 설정

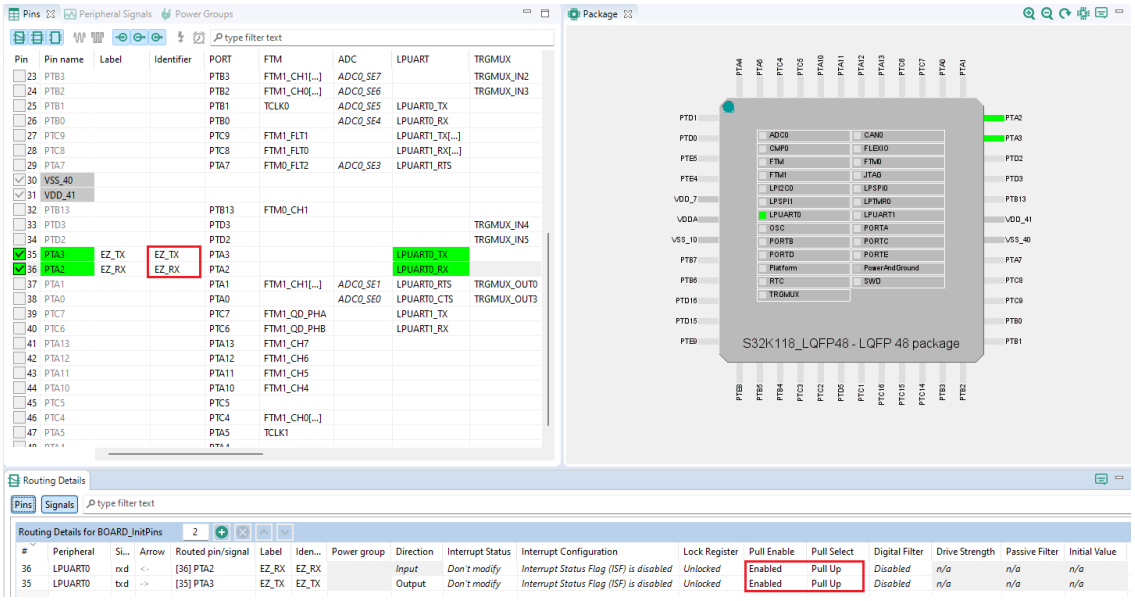
먼저 사용자 보드 환경에 맞춰 easyDSP, MCU 간 통신에 사용될 UART 채널 및 핀을 설정합니다. 채널 및 핀 사용에 제한사항은 없습니다.
만약 easyDSP 가 플래시 프로그래밍을 수행하지 않을 경우라면, /BOOT 와 /RESET 은 연결할 필요가 없습니다.



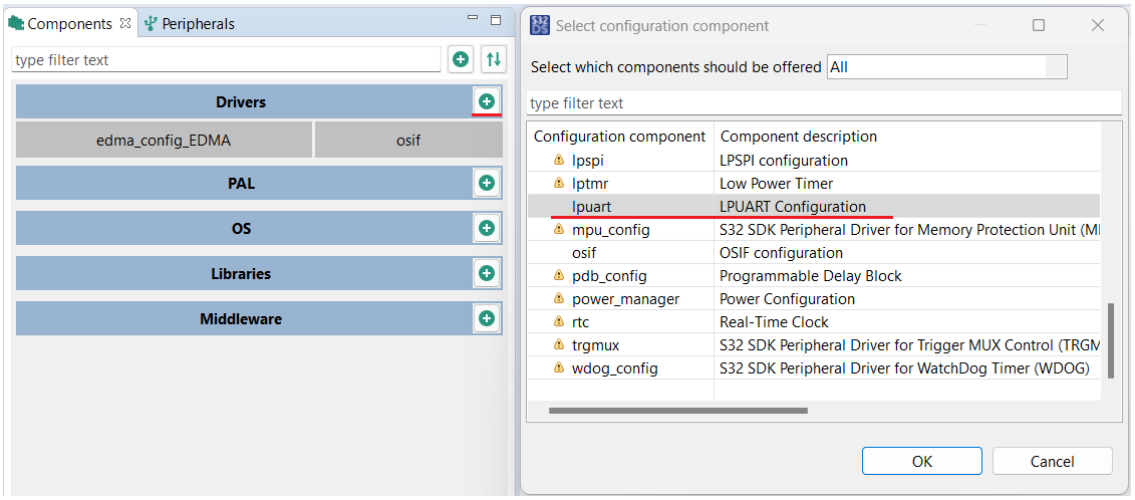
기타 주의 사항 :
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU /RESET 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 0.5 초내에 신호를 전달해야 함.
- easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업되어 있습니다.

STEP 2 : S32 Configuration Tools 설정

먼저 사용자 보드 환경에 맞춰 easyDSP 에서 사용할 UART 채널 및 핀을 설정합니다. 앞서 하드웨어 설정과 일치되어야 합니다.
TX, RX 핀의 Identifier 에 각각 EZ_TX, EZ_RX 를 입력합니다.
아래 그림에서는 LPUART0 을 사용하며 RX 는 PTA2, TX 는 PTA3 를 사용할 경우입니다.
각 핀의 성질을 Rounting Details 탭과 같이 설정합니다. 특히 풀업을 설정하는 것에 주의하세요.



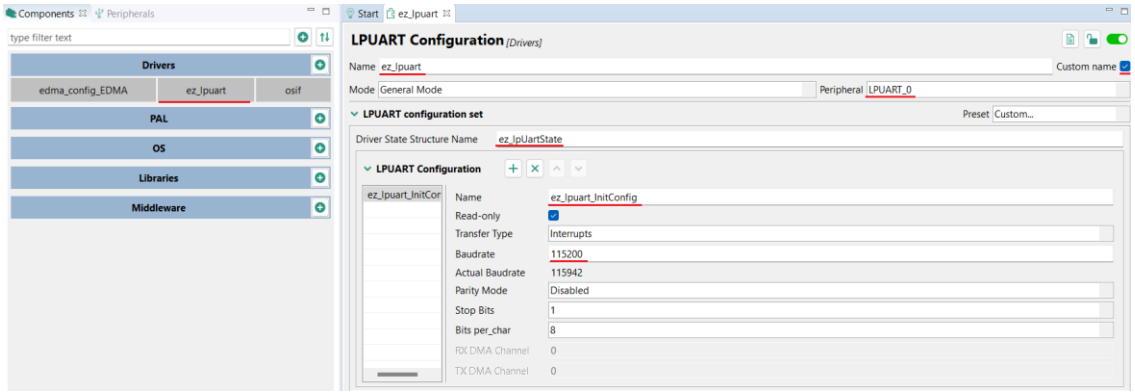
그리고 Drivers 에 lpuart 모듈을 추가합니다.



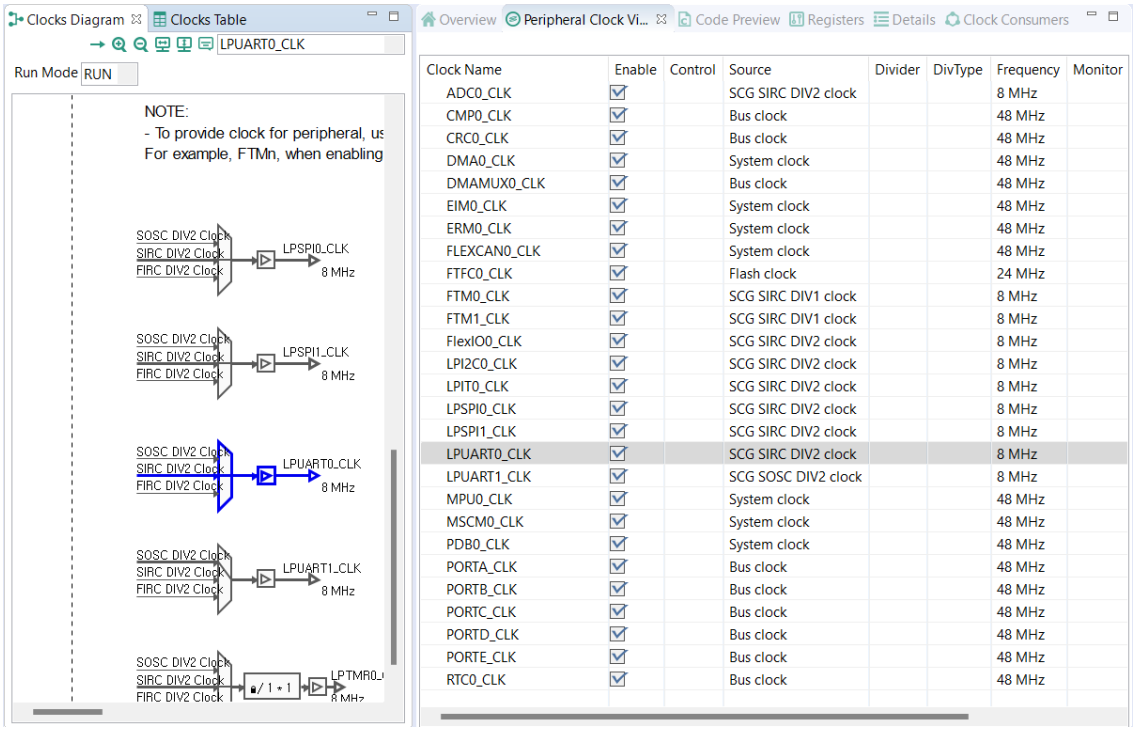
추가된 모듈에 대해서 설정합니다.

이름을 하기 그림과 같이 설정하며, UART 사용 채널을 선정합니다. 여기서는 앞서 그림 예제와 같이 LPUART0 을 사용합니다.

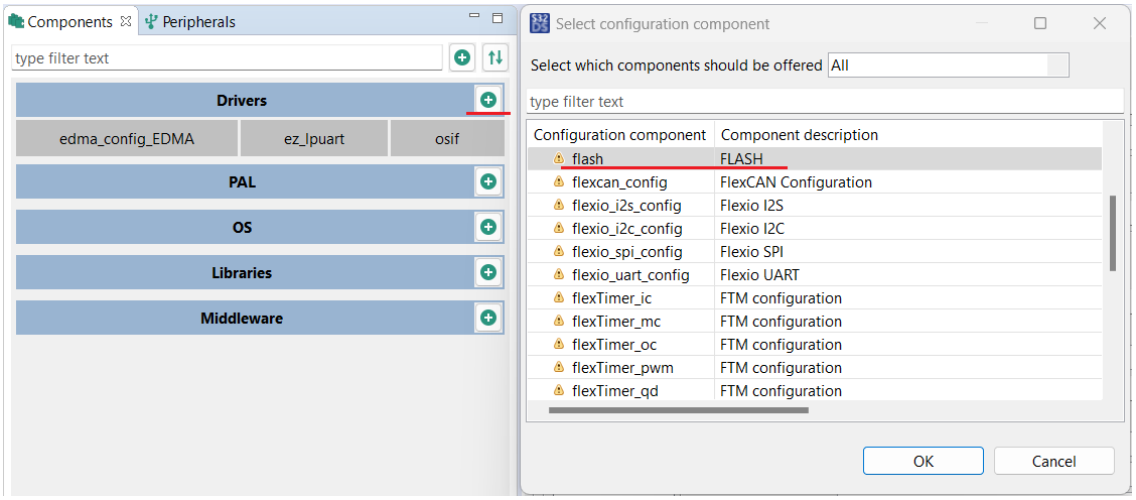
또한 각종 설정치를 그림과 같이 설정합니다. Baudrate 설정치는 easyDSP 프로젝트에서 사용될 보드레이트와 동일하게 설정합니다.

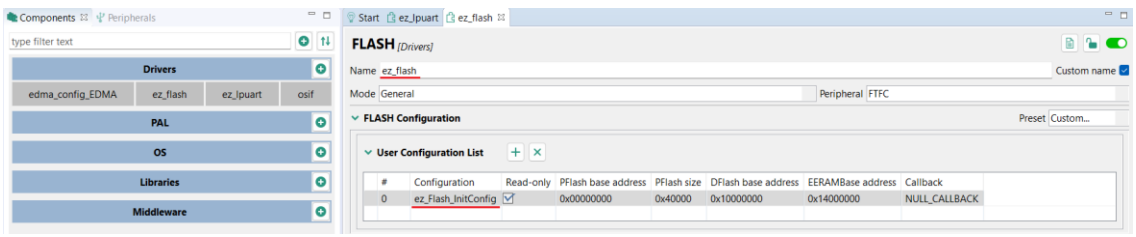


또한 해당 UART 채널에 적절한 클럭이 공급될 수 있도록 설정하여 주세요. 아래 예제 참조하세요.



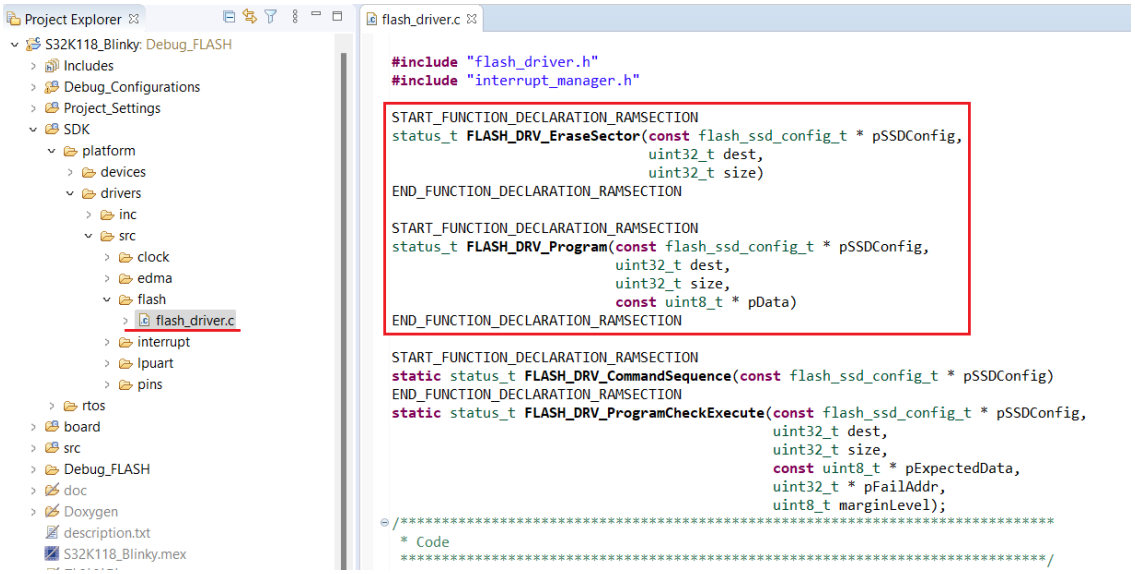
여기까지는 변수 모니터링을 위한 설정이었습니다. 만약 플래시 프로그래밍을 위해 easyDSP 부트로더를 사용한다면, 하기 작업이 추가로 필요하게 됩니다. easyDSP 부트로더는 Flash driver 를 사용하므로 Drivers 에서 flash 를 추가하고, 해당 이름을 아래와 같이 설정해주세요.





STEP 3 : easyDSP 가 플래시 프로그래밍을 지원하는 경우 소스 코드 수정

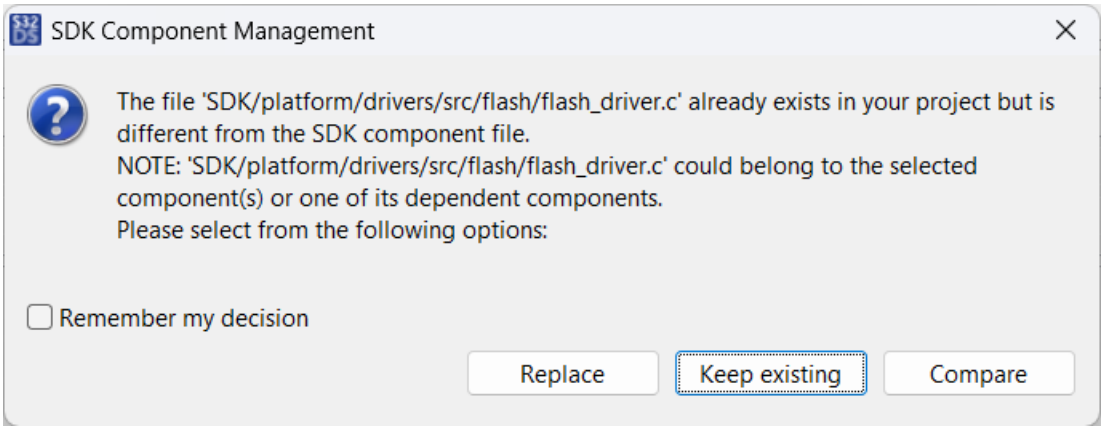
앞서 진행된 Configuration tool 설정 및 이로 인한 소스 코드 생성 이후, SDK > platform > drivers > src > flash 폴더내에 flash_driver.c 파일이 생성됩니다.
파일내 여러 함수 중 2 가지 함수를 easyDSP 가 사용하는데, 이 함수가 램에서 동작하도록 설정해줘야 합니다.
따라서 파일 서두에 하기 빨간색 박스를 추가하여, 함수를 선언해 주시고,



파일 중간에 해당 함수가 정의되어 있는 부분 앞,뒤로 하기 매크로를 추가하여 주시기 바랍니다.

```
START_FUNCTION_DEFINITION_RAMSECTION
status_t FLASH_DRV_EraseSector(const flash_ssd_config_t * pSSDConfig,
                               uint32_t dest,
                               uint32_t size)
{
    .... // contents of this function
}
END_FUNCTION_DEFINITION_RAMSECTION


START_FUNCTION_DEFINITION_RAMSECTION
status_t FLASH_DRV_Program(const flash_ssd_config_t * pSSDConfig,
                           uint32_t dest,
                           uint32_t size,
                           const uint8_t * pData)
{
    .... // contents of this function
}
END_FUNCTION_DEFINITION_RAMSECTION
```

Configuration Tool 이 자동 생성한 소스를 수정한 것이므로, Configuration tools 이 이를 감지하여, 소스파일을 다시 원래로 되돌릴 지 문의할 경우 이를 허용하지 마십시요.
예를 들어 아래 메시지에서 Keep existing 버튼을 선택해야 합니다.



STEP 4 : easyDSP 제공 소스 파일 및 함수 호출

먼저 easyDSP 통신 및 플래시프로그래밍을 위해 제공되는 소스파일 (easyS32K1_SDK.h, easyS32K1_SDK_comm.c, easyS32K1_SDK_boot.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더 아래 ₩source₩S32 에서 찾을 수 있습니다.

만약 easyDSP 를 사용하여 플래시 프로그래밍을 수행할 경우, easyS32K1_SDK.h 파일의 아래 부분에서 EZ_BOOTLOADER_USE 를 1 로 설정하세요. easyDSP 로 플래시 프로그래밍을 하지 않는 경우에는 EZ_BOOTLOADER_USE 를 0 으로 설정합니다.

```
/*****************************************************************************
 In case you use the bootloader provided by easyDSP to program flash,
 define EZ_BOOTLOADER_USE as 1
*****************************************************************************/
#define EZ_BOOTLOADER_USE 1
```

main.c 상단에 easyS32K1_SDK.h 를 include 하여 주시고, main 함수에서 초기화 루틴 이후 easyDSP_init() 함수를 호출하시기 바랍니다. easyDSP_init() 함수에서는 easyDSP 통신을 위한 각종 설정을 수행합니다.
만약 easyDSP 가 플래시 프로그래밍을 지원하길 원하시면 클럭 및 핀이 설정된 바로 직후 easyDSP_boot()를 호출하시기 바랍니다.

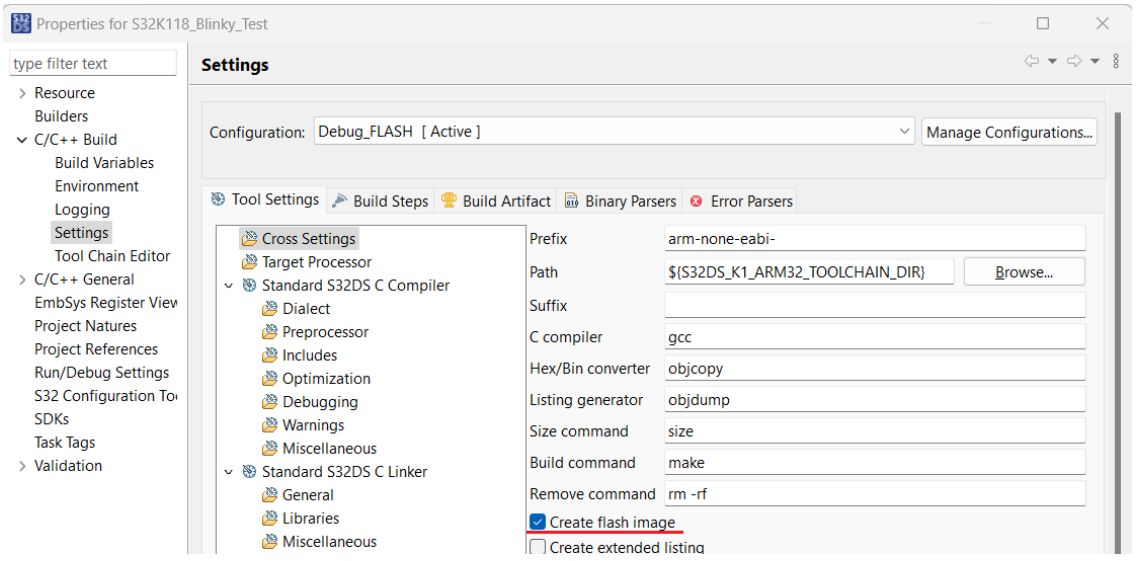
```
#include "easyS32K1_SDK.h"

int main(void)
{
    /* Initialize and configure clocks - see clock manager component for details */
    CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                   g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_AGREEMENT);

    /* Initialize pins - See PinSettings component for more info */
    PINS_DRV_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);

#if EZ_BOOTLOADER_USE
    // Right after clock and pin setting, call easyDSP_boot() to enable flash programming
    easyDSP_boot();
#endif

    // reset of initial setting
    .
    .
    .
    .
    .
    .

    // call easyDSP_init() to enable easyDSP monitoring
    easyDSP_init();

    // loop forever
    while(1)
    {
        .
        .
        .
    }
}
```

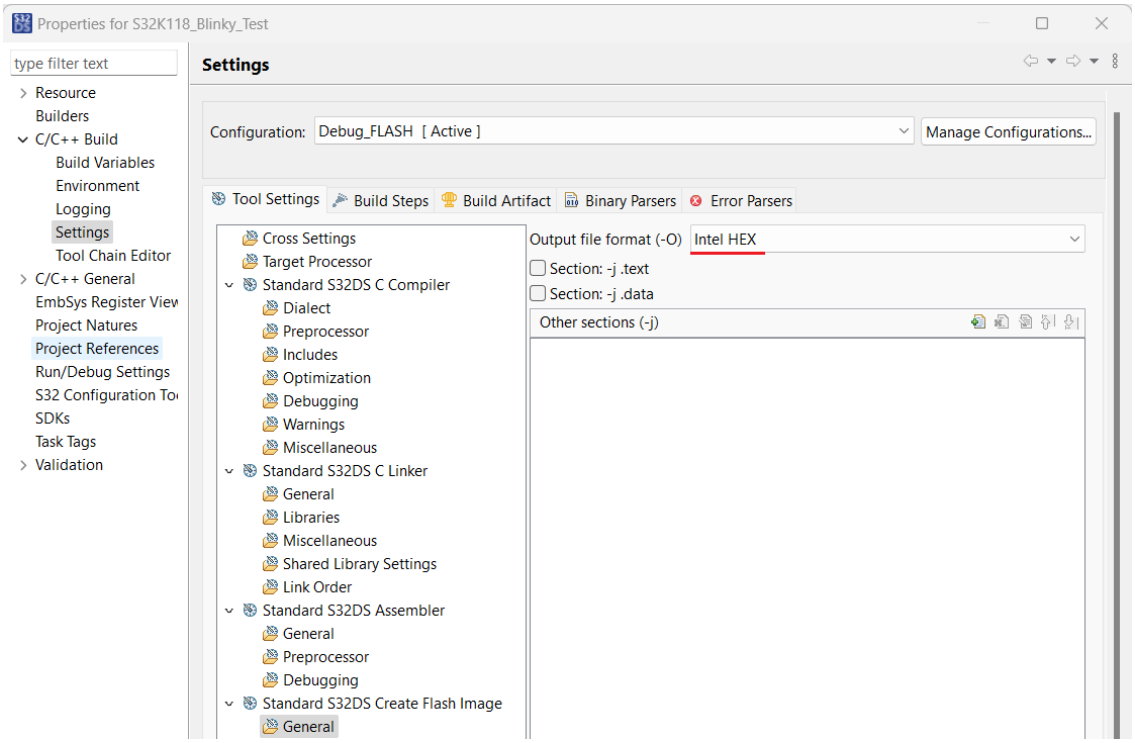
STEP 5 : IDE 설정

1. 매 컴파일마다 hex 파일 (인텔 형식) 이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 IDE 를 설정해주세요. Hex 파일은 플래시 프로그래밍할 때 사용됩니다.

Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.

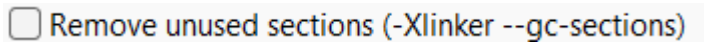
S32DS 기준 하기 옵션 설정 참조하세요.





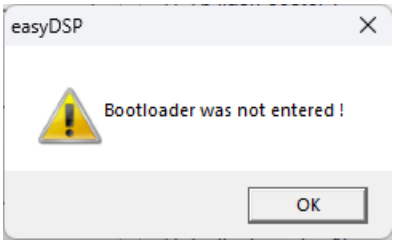
2. easyDSP 로 모니터링을 수행하기 위해서는, 출력 파일 (예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 해당 변수를 모니터링하려면 적절한 IDE 세팅이 필요합니다. 예를 들어, S32DS 의 프로젝트 설정 > 링커 옵션 하기 항목을 체크하지 않습니다.



STEP 6 : easyDSP 부트로더의 제한 사항

1. S32K1 MCU 는 제작업체에서 제공하는 롬부트로더가 없기에 사용자가 부트로더를 별도로 제작 사용해야 합니다. easyDSP 는 사용자 프로그램 안에 easyDSP 가 제공하는 부트로더 (함수 easyDSP_boot)가 존재하는 구성입니다. 따라서 MCU 플래시에 easyDSP 소스파일이 이미 프로그래밍되어 있어야 새로운 프로그램으로 재 프로그래밍이 가능합니다. 만약 플래시가 전부 지워져 있거나, easyDSP 소스파일이 플래시에 프로그래밍이 되어 있지 않다면, 플래시 프로그래밍이 지원되지 않으며 하기와 같은 메시지가 송출되며, 이 경우 디버거를 사용한 플래시 프로그래밍이 필요합니다. 따라서 처음 한번은 디버거를 이용해서 플래시 프로그래밍을 해야만 합니다.



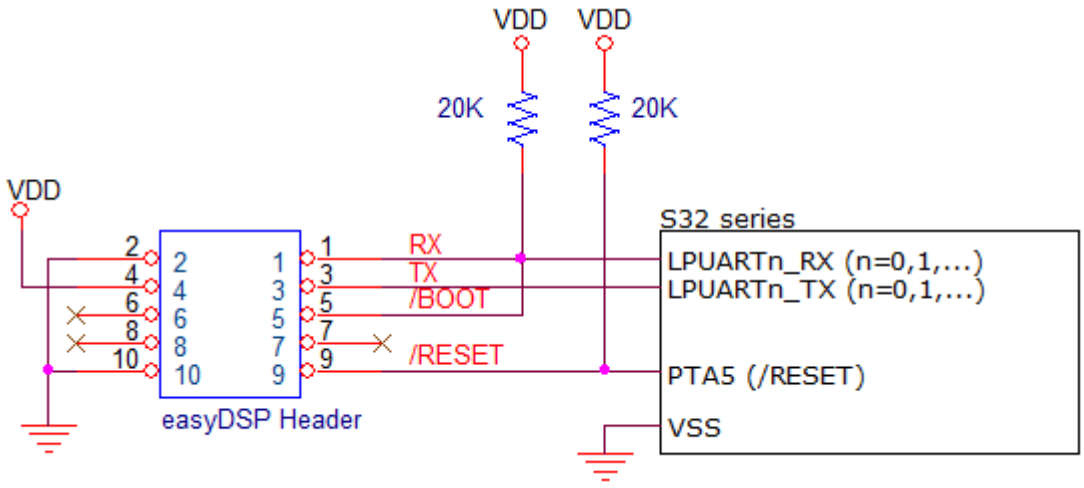
2. easyDSP 부트로더는 램에서 동작해야 하기에 약 1.25K 바이트 램 영역을 소비합니다 (-O1 최적화 옵션 기준).

7.3.2 S32K/S32M + RTD 사용

본 페이지는 사용자가 S32 Configuration Tools 및 RTD(Real-Time Drivers)를 사용함을 전제하고 있습니다.

STEP 1 : 하드웨어 설정

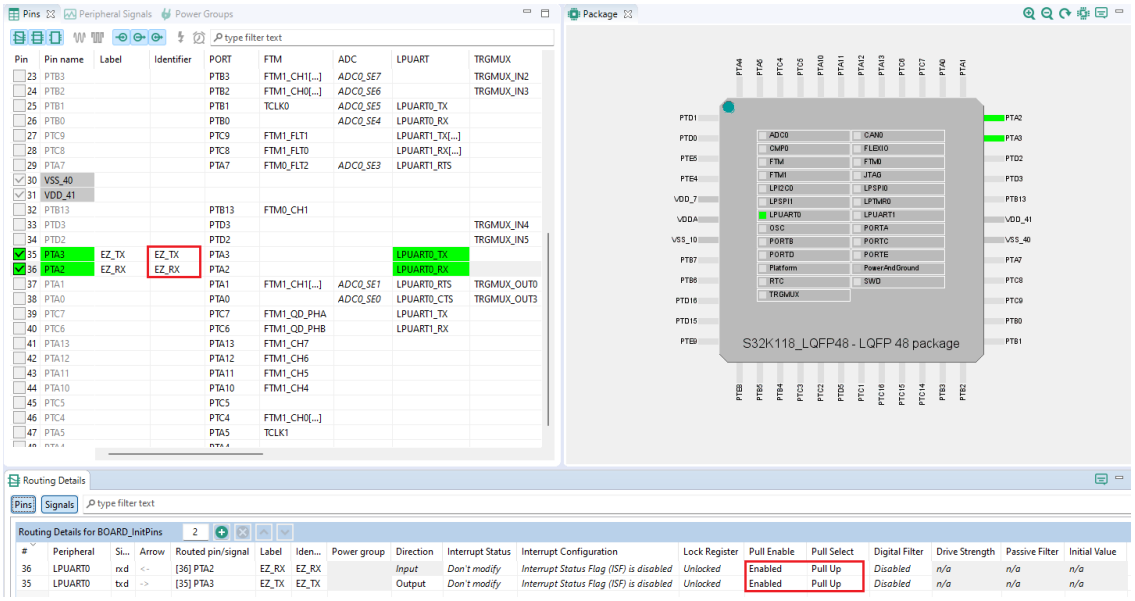
먼저 사용자 보드 환경에 맞춰 easyDSP, MCU 간 통신에 사용될 UART 채널 및 핀을 설정합니다.
채널 및 핀 사용에 제한 사항은 없습니다. 단, S32M 의 경우 LPUART1 은 사용할 수 없습니다.
만약 easyDSP 가 플래시 프로그래밍을 수행하지 않을 경우라면, /BOOT 와 /RESET 은 연결할 필요가 없습니다.



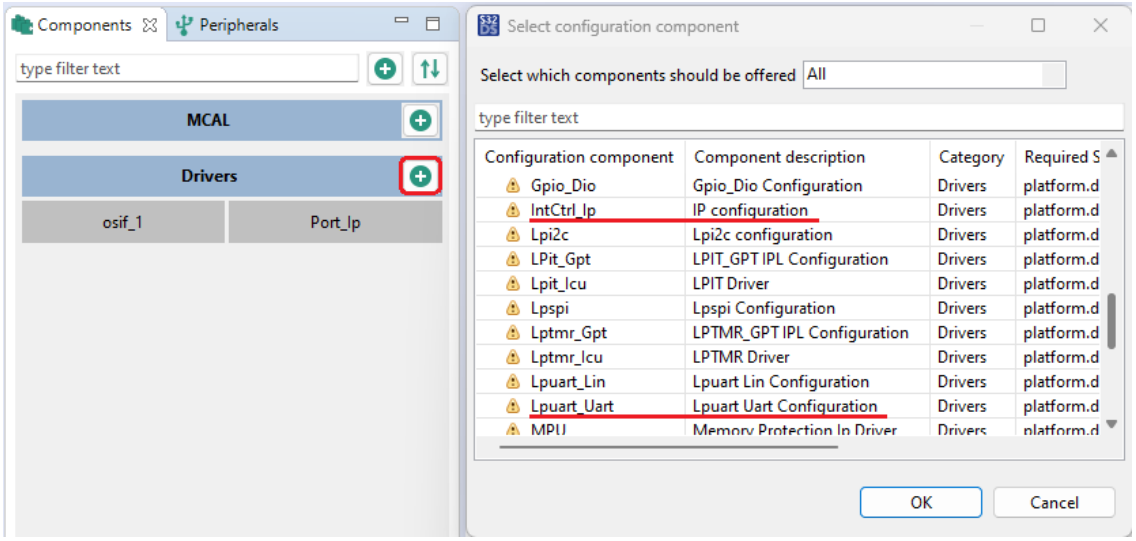
기타 주의 사항 :
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU /RESET 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 0.5 초내에 신호를 전달해야 함.
- easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업되어 있습니다.

STEP 2 : easyDSP 모니터링을 위한 S32 Configuration Tools 설정

먼저 사용자 보드 환경에 맞춰 easyDSP 에서 사용할 UART 채널 및 핀을 설정합니다. 앞서 하드웨어 설정과 일치되어야 합니다.
TX, RX 핀의 Identifier 에 각각 EZ_TX, EZ_RX 를 입력합니다.
아래 그림에서는 LPUART0 을 사용하며 RX 는 PTA2, TX 는 PTA3 를 사용할 경우입니다.
각 핀의 성질을 Rounting Details 탭과 같이 설정합니다. 특히 풀업을 설정하는 것에 주의하세요.



그리고 Drivers 에 Lpuart_Uart, IntCtrl_Ip 모듈을 추가합니다. 이미 해당 모듈이 있다면 추가없이 사용하시면 됩니다.

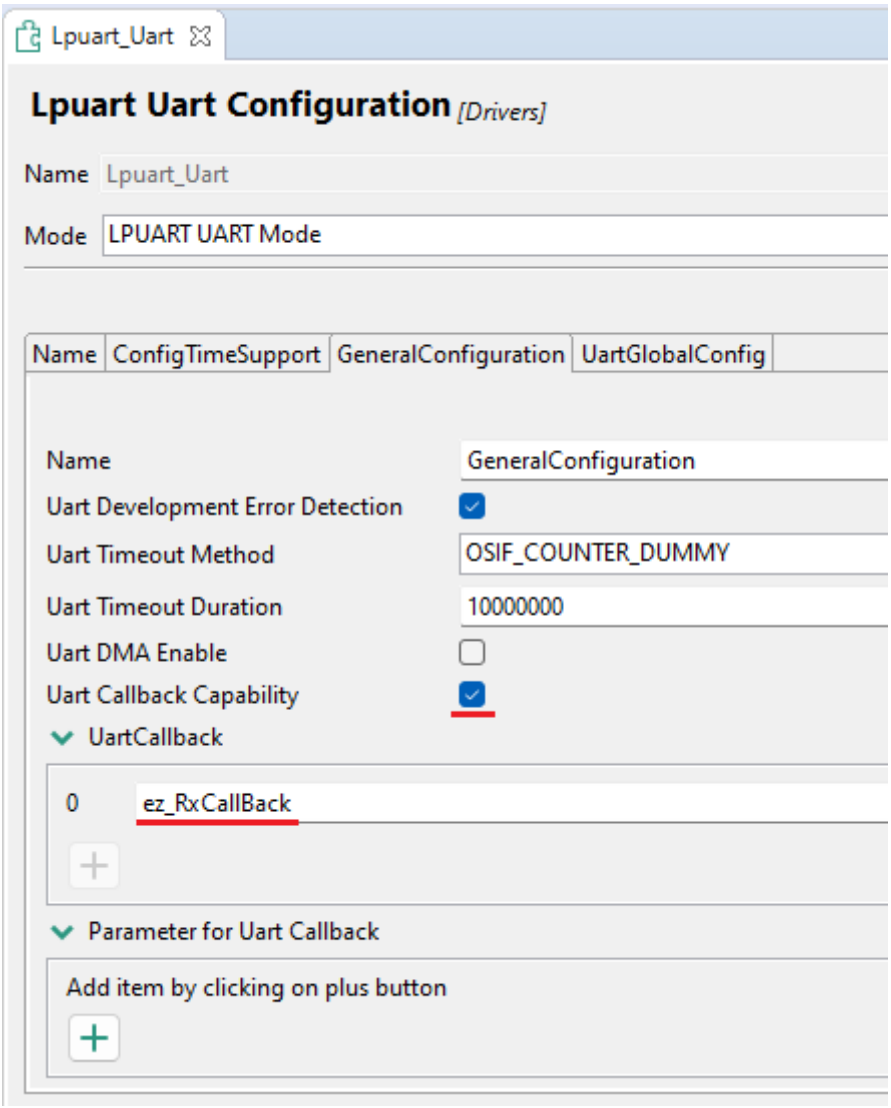


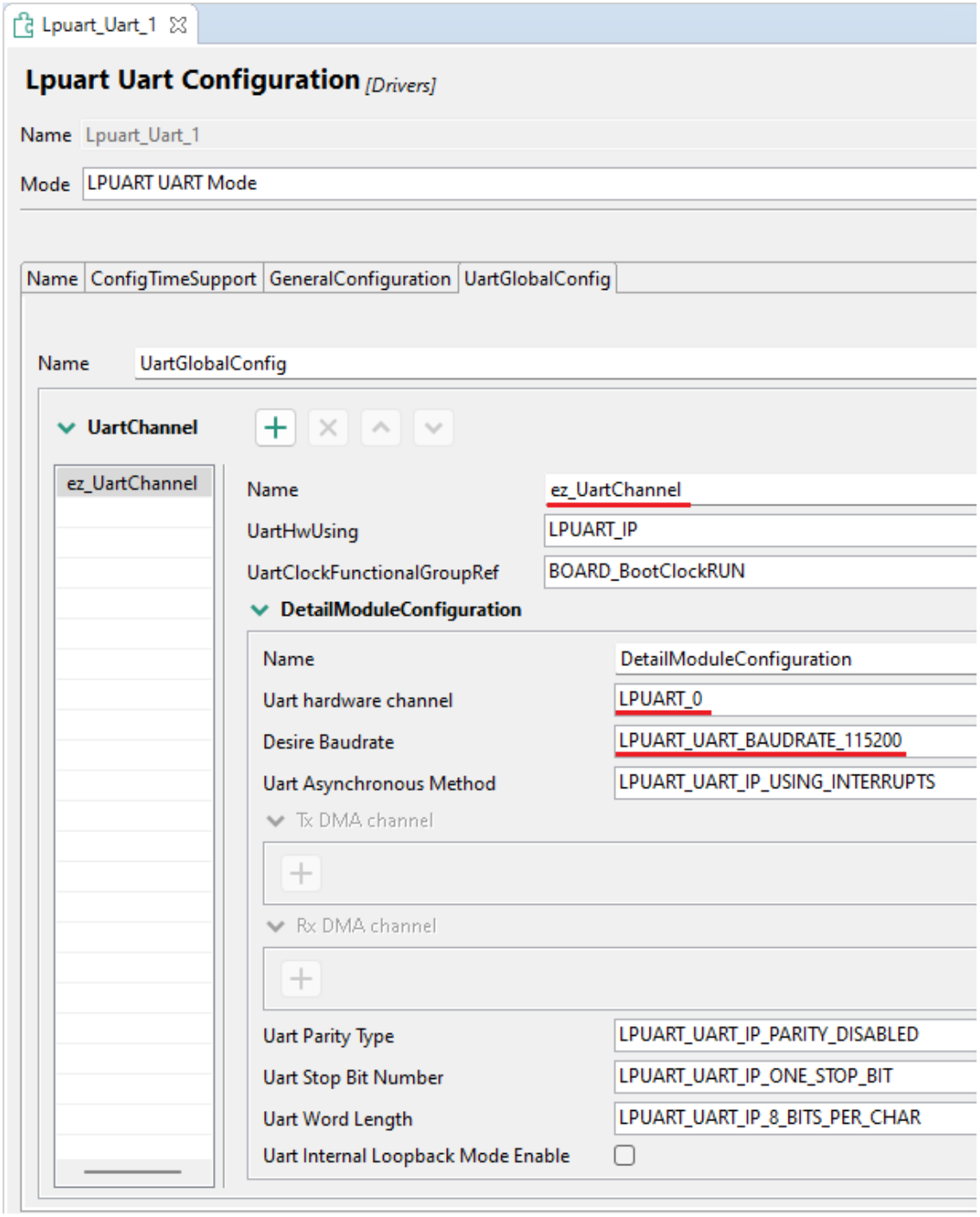
Lpuart_Uart 모듈에 대해서 설정합니다.

GeneralConfiguration 탭에서 Uart Callback Capability 를 활성화하고, 그 이름을 ez_RxCallBack 으로 설정합니다.

UartGlobalConfig 탭에서 이름을 하기와 같이 지정하시고, UART 사용 채널을 선정합니다. 여기서는 앞서 스텝 1, 2 와 같이 LPUART0 을 사용합니다.

또한 각종 설정치를 그림과 같이 설정합니다. Baudrate 설정치는 easyDSP 프로젝트에서 사용될 보드레이트와 동일하게 설정합니다.


Lpuart_Uart
Lpuart Uart Configuration [Drivers]
Name Lpuart_Uart
Mode LPUART UART Mode
Name
ConfigTimeSupport
GeneralConfiguration
UartGlobalConfig
Name GeneralConfiguration
Uart Development Error Detection
Uart Timeout Method OSIF_COUNTER_DUMMY
Uart Timeout Duration 10000000
Uart DMA Enable
Uart Callback Capability
UartCallback
0 ez_RxCallBack
Parameter for Uart Callback
Add item by clicking on plus button



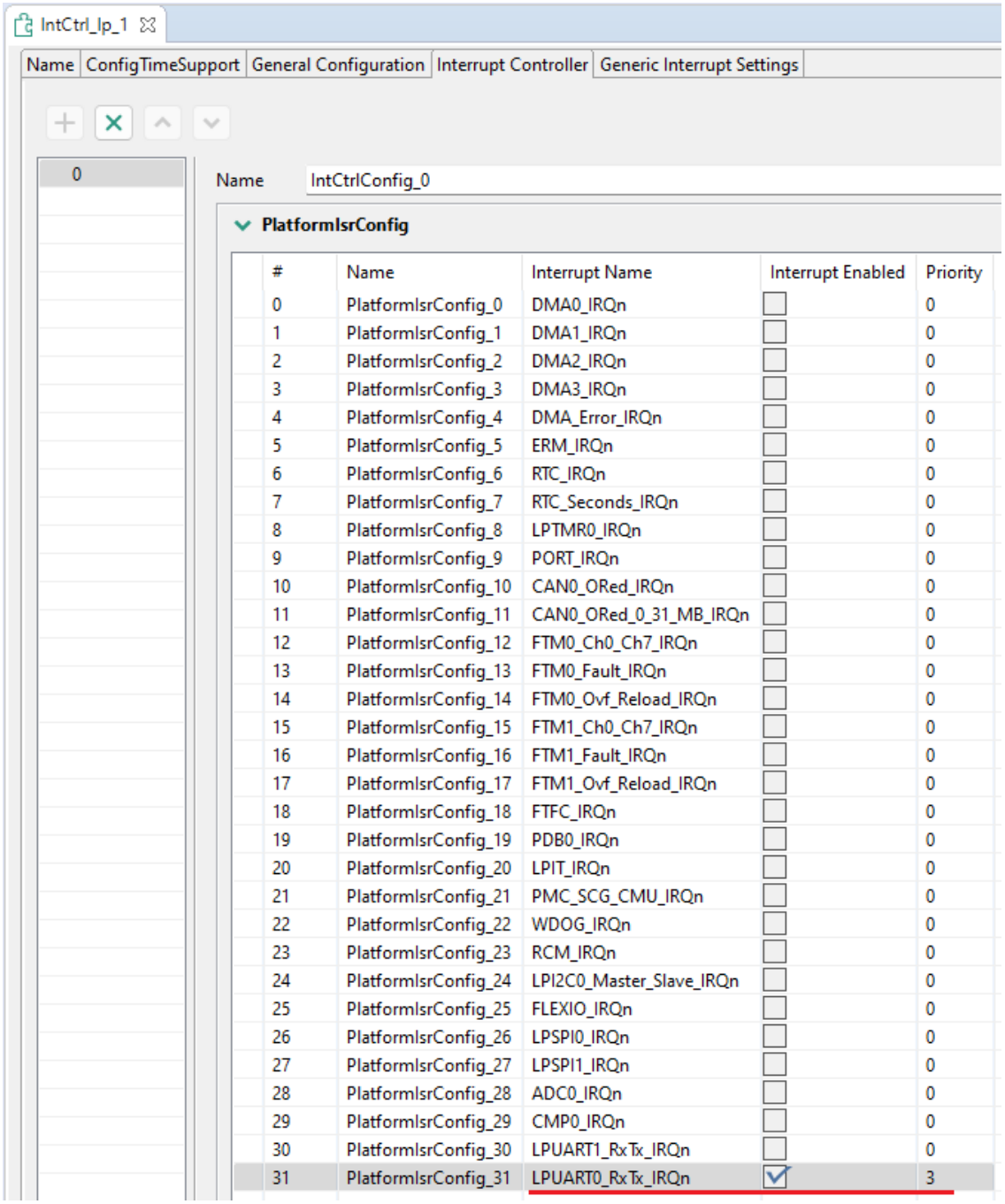
IntCtrl_Ip 모듈에 대해서 설정합니다.

Interrupt Controller 탭에서 easyDSP 통신에 사용할 LPUART 채널의 인터럽트를 활성화시키고 그 순위를 최하위(가장 큰 값)로 설정합니다.

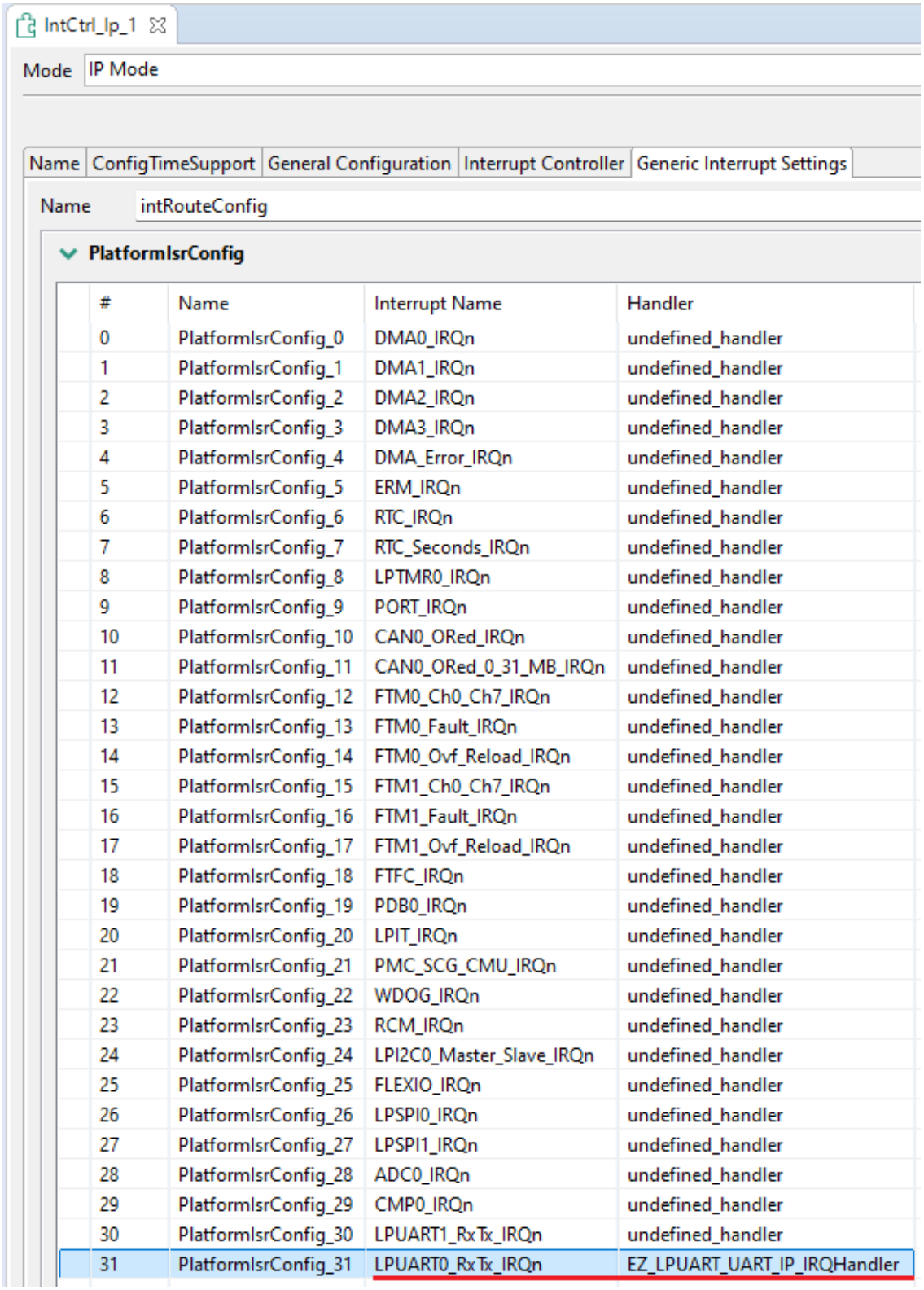
Generic Interrupt Setting 탭에서 해당 LPUART 채널 인터럽트의 인터럽트 핸들러를 EZ_LPUART_UART_IP_IRQHandler 로 설정합니다.

MCU 종류에 따라 두 탭의 역할이 하나의 탭으로 합쳐져 있는 경우도 있습니다.

아래 그림에서는 앞서 예제와 같이 LPUART0 을 사용합니다.

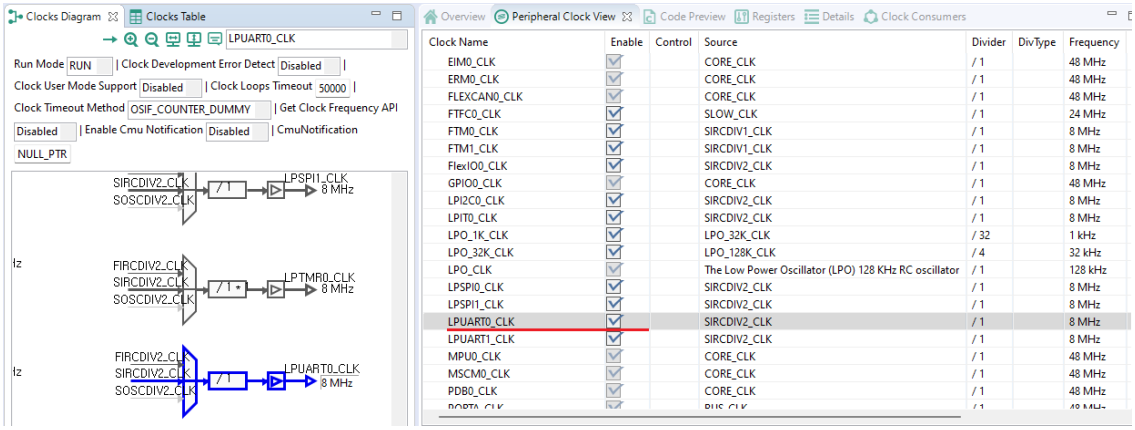


#	Name	Interrupt Name	Interrupt Enabled	Priority
0	PlatformIsrConfig_0	DMA0_IRQn	☐	0
1	PlatformIsrConfig_1	DMA1_IRQn	☐	0
2	PlatformIsrConfig_2	DMA2_IRQn	☐	0
3	PlatformIsrConfig_3	DMA3_IRQn	☐	0
4	PlatformIsrConfig_4	DMA_Error_IRQn	☐	0
5	PlatformIsrConfig_5	ERM_IRQn	☐	0
6	PlatformIsrConfig_6	RTC_IRQn	☐	0
7	PlatformIsrConfig_7	RTC_Seconds_IRQn	☐	0
8	PlatformIsrConfig_8	LPTMR0_IRQn	☐	0
9	PlatformIsrConfig_9	PORT_IRQn	☐	0
10	PlatformIsrConfig_10	CAN0_ORed_IRQn	☐	0
11	PlatformIsrConfig_11	CAN0_ORed_0_31_MB_IRQn	☐	0
12	PlatformIsrConfig_12	FTM0_Ch0_Ch7_IRQn	☐	0
13	PlatformIsrConfig_13	FTM0_Fault_IRQn	☐	0
14	PlatformIsrConfig_14	FTM0_Ovf_Reload_IRQn	☐	0
15	PlatformIsrConfig_15	FTM1_Ch0_Ch7_IRQn	☐	0
16	PlatformIsrConfig_16	FTM1_Fault_IRQn	☐	0
17	PlatformIsrConfig_17	FTM1_Ovf_Reload_IRQn	☐	0
18	PlatformIsrConfig_18	FTFC_IRQn	☐	0
19	PlatformIsrConfig_19	PDB0_IRQn	☐	0
20	PlatformIsrConfig_20	LPIT_IRQn	☐	0
21	PlatformIsrConfig_21	PMC_SCG_CMU_IRQn	☐	0
22	PlatformIsrConfig_22	WDOG_IRQn	☐	0
23	PlatformIsrConfig_23	RCM_IRQn	☐	0
24	PlatformIsrConfig_24	LPI2C0_Master_Slave_IRQn	☐	0
25	PlatformIsrConfig_25	FLEXIO_IRQn	☐	0
26	PlatformIsrConfig_26	LPSPI0_IRQn	☐	0
27	PlatformIsrConfig_27	LPSPI1_IRQn	☐	0
28	PlatformIsrConfig_28	ADC0_IRQn	☐	0
29	PlatformIsrConfig_29	CMP0_IRQn	☐	0
30	PlatformIsrConfig_30	LPUART1_RxTx_IRQn	☐	0
31	PlatformIsrConfig_31	LPUART0_RxTx_IRQn	☑	3



#	Name	Interrupt Name	Handler
0	PlatformIsrConfig_0	DMA0_IRQn	undefined_handler
1	PlatformIsrConfig_1	DMA1_IRQn	undefined_handler
2	PlatformIsrConfig_2	DMA2_IRQn	undefined_handler
3	PlatformIsrConfig_3	DMA3_IRQn	undefined_handler
4	PlatformIsrConfig_4	DMA_Error_IRQn	undefined_handler
5	PlatformIsrConfig_5	ERM_IRQn	undefined_handler
6	PlatformIsrConfig_6	RTC_IRQn	undefined_handler
7	PlatformIsrConfig_7	RTC_Seconds_IRQn	undefined_handler
8	PlatformIsrConfig_8	LPTMR0_IRQn	undefined_handler
9	PlatformIsrConfig_9	PORT_IRQn	undefined_handler
10	PlatformIsrConfig_10	CAN0_ORed_IRQn	undefined_handler
11	PlatformIsrConfig_11	CAN0_ORed_0_31_MB_IRQn	undefined_handler
12	PlatformIsrConfig_12	FTM0_Ch0_Ch7_IRQn	undefined_handler
13	PlatformIsrConfig_13	FTM0_Fault_IRQn	undefined_handler
14	PlatformIsrConfig_14	FTM0_Ovf_Reload_IRQn	undefined_handler
15	PlatformIsrConfig_15	FTM1_Ch0_Ch7_IRQn	undefined_handler
16	PlatformIsrConfig_16	FTM1_Fault_IRQn	undefined_handler
17	PlatformIsrConfig_17	FTM1_Ovf_Reload_IRQn	undefined_handler
18	PlatformIsrConfig_18	FTFC_IRQn	undefined_handler
19	PlatformIsrConfig_19	PDB0_IRQn	undefined_handler
20	PlatformIsrConfig_20	LPIT_IRQn	undefined_handler
21	PlatformIsrConfig_21	PMC_SCG_CMU_IRQn	undefined_handler
22	PlatformIsrConfig_22	WDOG_IRQn	undefined_handler
23	PlatformIsrConfig_23	RCM_IRQn	undefined_handler
24	PlatformIsrConfig_24	LPI2C0_Master_Slave_IRQn	undefined_handler
25	PlatformIsrConfig_25	FLEXIO_IRQn	undefined_handler
26	PlatformIsrConfig_26	LPSPI0_IRQn	undefined_handler
27	PlatformIsrConfig_27	LPSPI1_IRQn	undefined_handler
28	PlatformIsrConfig_28	ADC0_IRQn	undefined_handler
29	PlatformIsrConfig_29	CMP0_IRQn	undefined_handler
30	PlatformIsrConfig_30	LPUART1_RxTx_IRQn	undefined_handler
31	PlatformIsrConfig_31	LPUART0_RxTx_IRQn	EZ_LPUART_UART_IP_IRQHandler

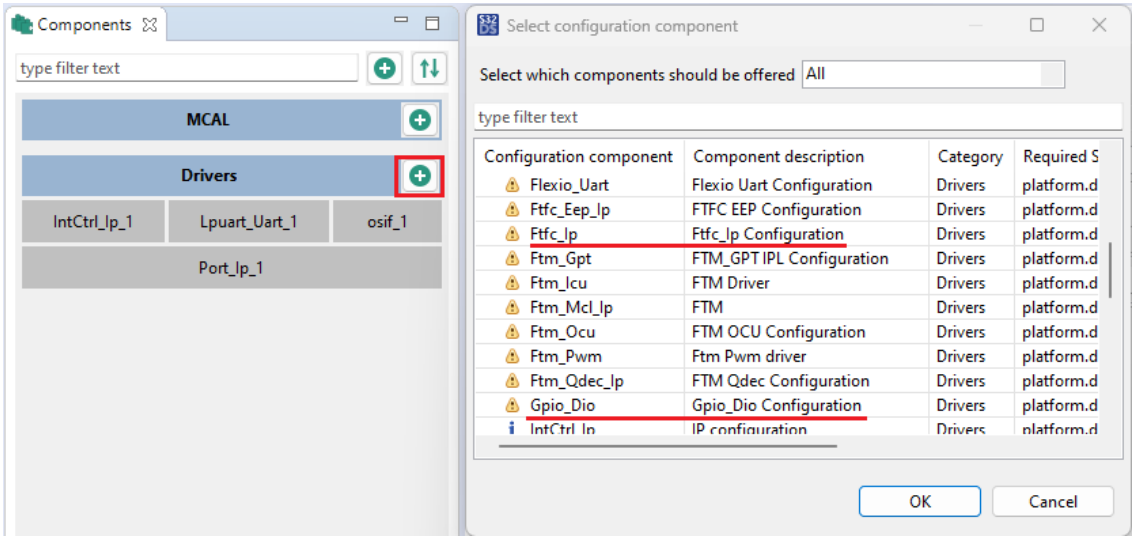
또한 해당 UART 채널에 적절한 클럭이 공급될 수 있도록 설정하여 주세요. 아래 예제 참조하세요.



STEP 3-1 : S32K1x 에서 easyDSP 부트로더를 위한 S32 Configuration Tools 설정

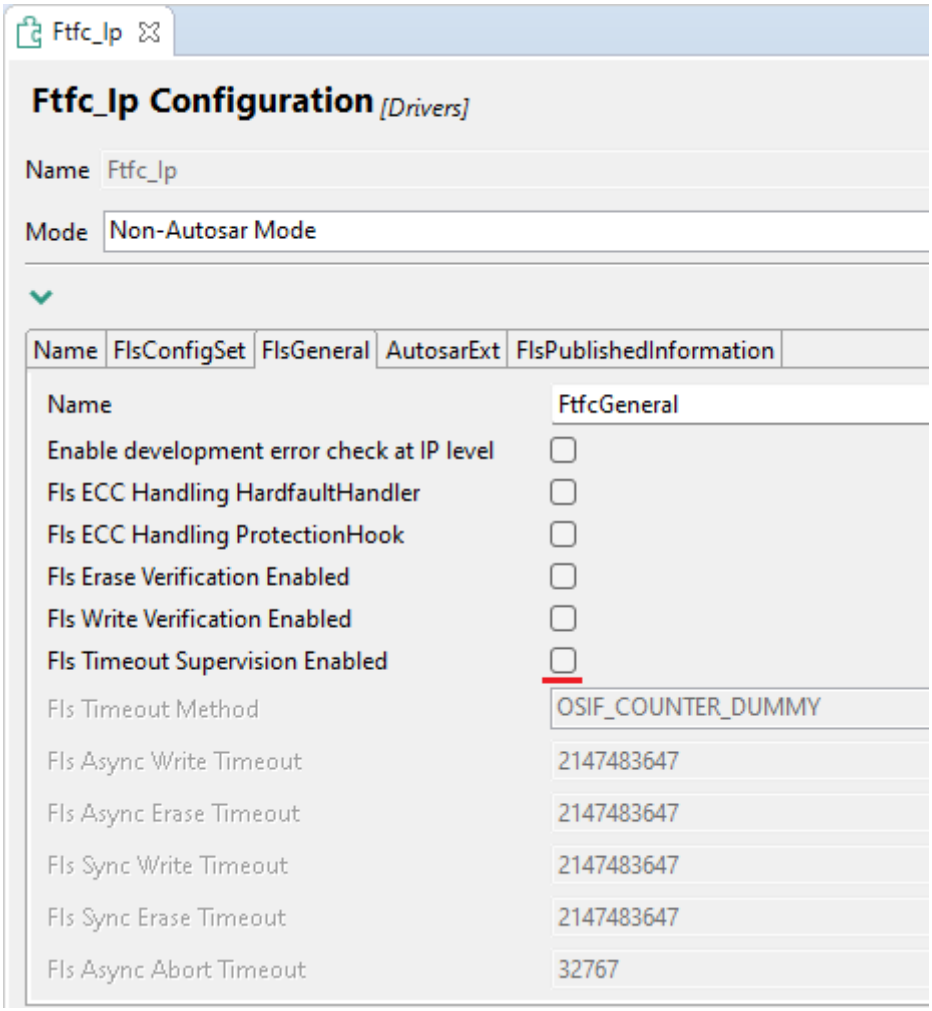
만약 S32K1x 에서 easyDSP 부트로더를 사용하여 플래시 프로그래밍을 수행할 경우, 하기 작업이 추가로 필요하게 됩니다.

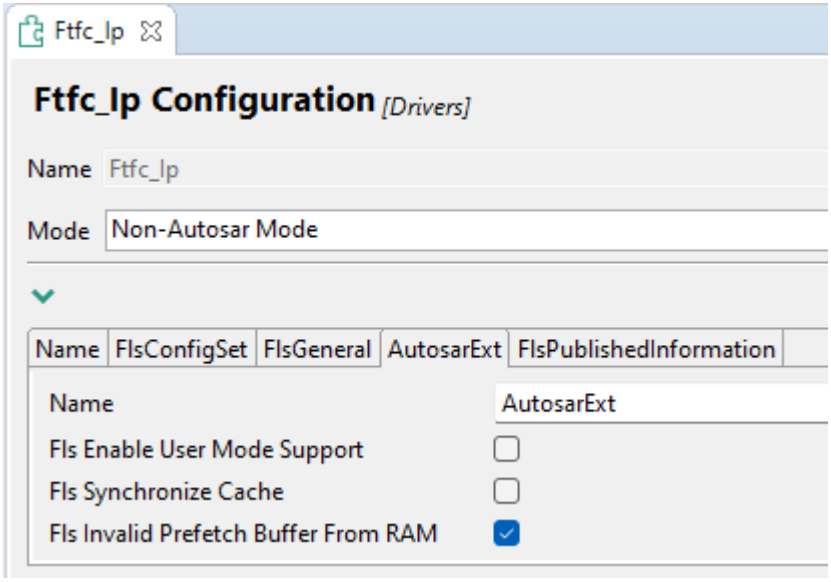
easyDSP 부트로더가 사용할 Ftfc_lp 모듈 및 Gpio_Dio 모듈을 추가합니다.



Gpio_Dio 모듈은 별도의 설정이 필요 없습니다.

Ftfc_lp 모듈의 FlsGeneral 탭에서 Fls Timeout Supervision Enabled 버튼을 비활성화합니다.

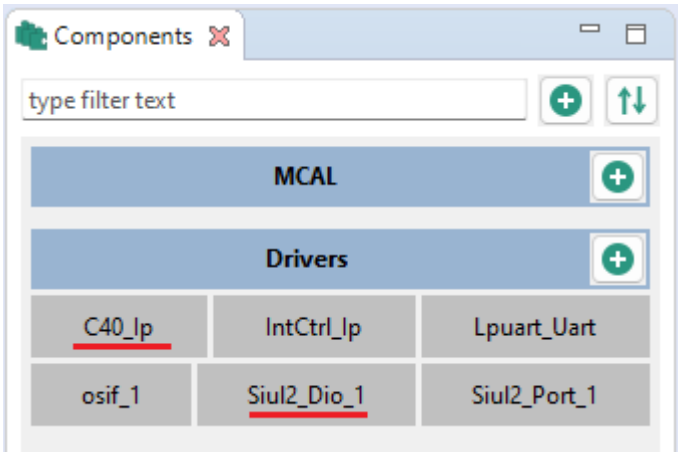




STEP 3-2 : S32K3x 에서 easyDSP 부트로더를 위한 S32 Configuration Tools 설정

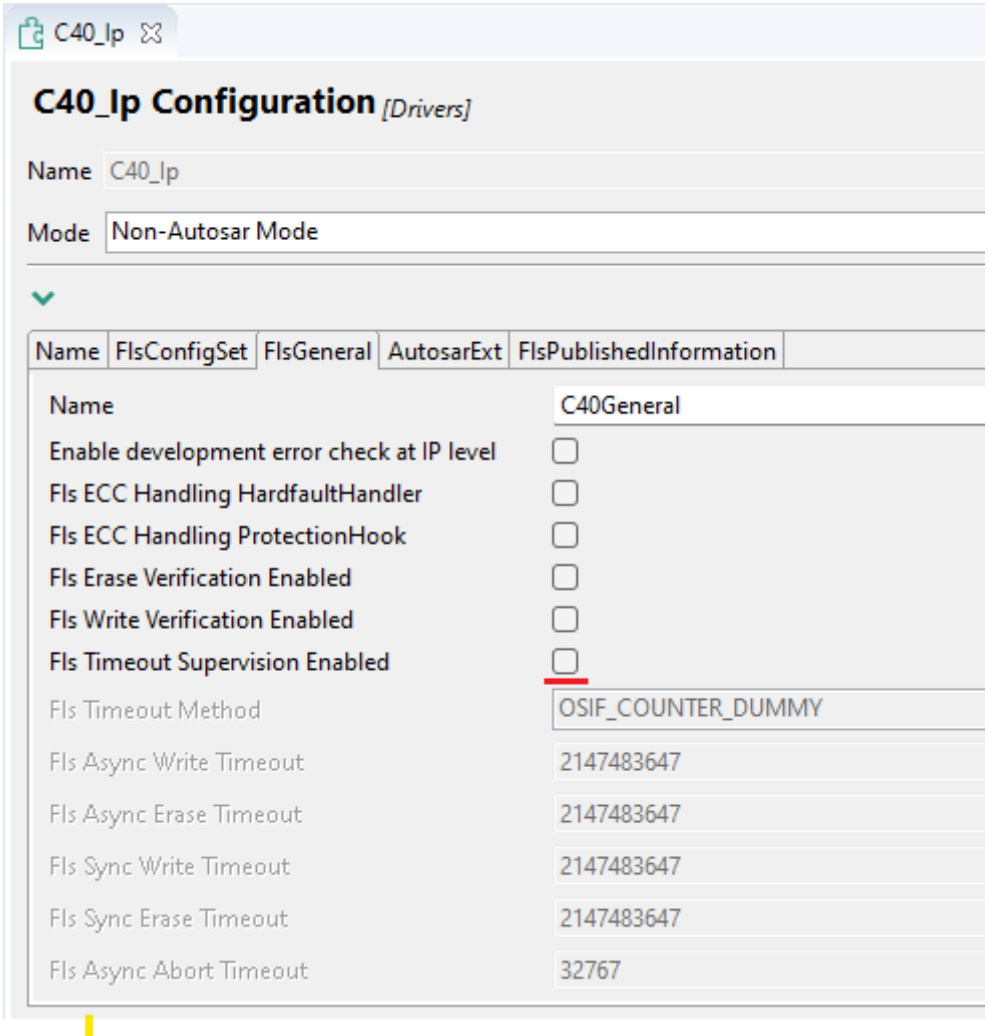
만약 S32K3x 에서 easyDSP 부트로더를 사용하여 플래시 프로그래밍을 수행할 경우, 하기 작업이 추가로 필요하게 됩니다.

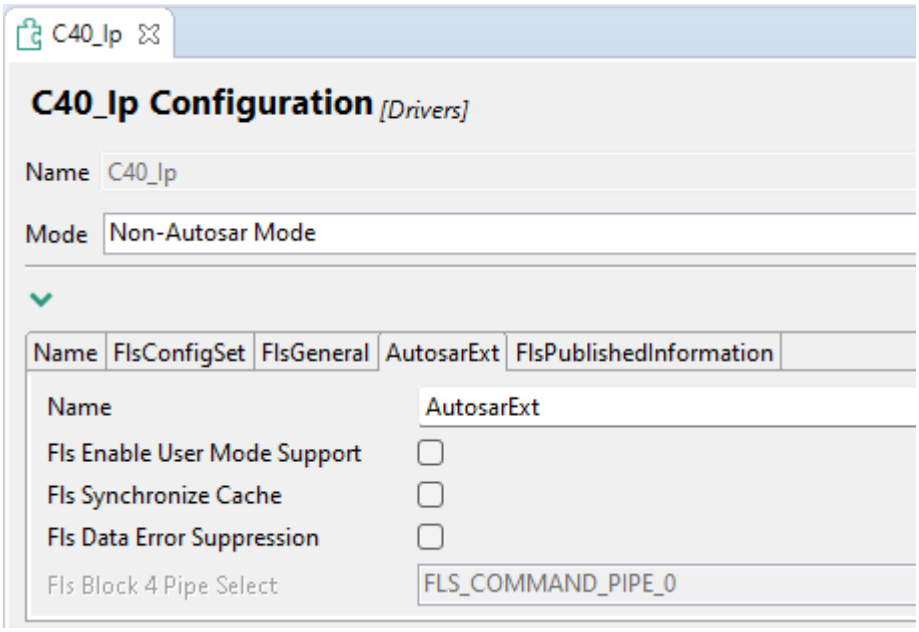
easyDSP 부트로더가 사용할 C40_Ip 모듈 및 Siu2_Dio 모듈을 추가합니다.



Siu2_Dio 모듈은 별도의 설정이 필요 없습니다.

C40_Ip 모듈의 FlsGeneral 탭에서 Fls Timeout Supervision Enabled 버튼을 비활성화합니다.

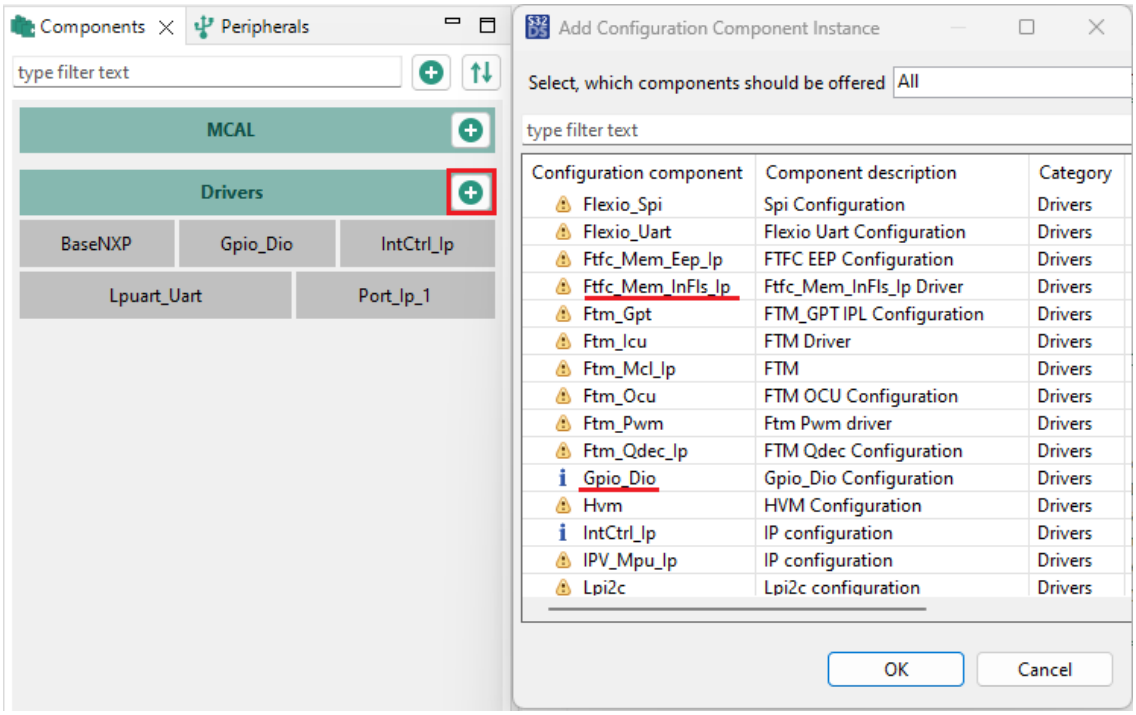




STEP 3-3 : S32M24x 에서 easyDSP 부트로더를 위한 S32 Configuration Tools 설정

만약 S32M24x 에서 easyDSP 부트로더를 사용하여 플래시 프로그래밍을 수행할 경우, 하기 작업이 추가로 필요하게 됩니다.

easyDSP 부트로더가 사용할 Ftfc_Mem_InFls_Ip 모듈 및 Gpio_Dio 모듈을 추가합니다.



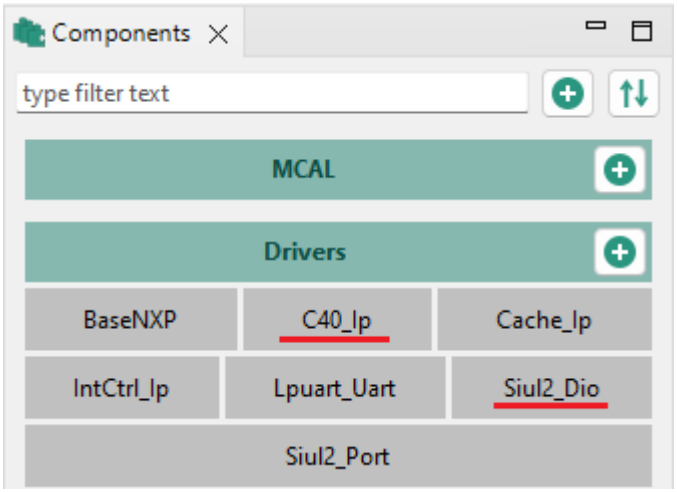
Gpio_Dio 모듈은 별도의 설정이 필요 없습니다.

Ftfc_Mem_InFls_Ip 모듈도 별도 설정없이 기본 설정 사용합니다. MemGeneral 탭에서 Mem Timeout Supervision Enabled 버튼 및 MemAutosarExt 탭의 Mem Synchronize Cache 버튼은 비활성화되어야 함을 주의하세요.

STEP 3-4 : S32M27x 에서 easyDSP 부트로더를 위한 S32 Configuration Tools 설정

만약 S32M27x 에서 easyDSP 부트로더를 사용하여 플래시 프로그래밍을 수행할 경우, 하기 작업이 추가로 필요하게 됩니다.

easyDSP 부트로더가 사용할 C40_Ip 모듈 및 Siu2_Dio 모듈을 추가합니다.



Siu2_Dio 모듈은 별도의 설정이 필요 없습니다.

C40_Ip 모듈도 별도 설정없이 기본 설정 사용합니다. MemGeneral 탭에서 Mem Timeout Supervision Enabled 버튼 및 MemAutosarExt 탭의 Mem Synchronize Cache 버튼은 비활성화되어야 함을 주의하세요.

STEP 4-1 : S32K1x 에서 easyDSP 부트로더를 위한 소스 코드 수정

S32K1x 에서 easyDSP 부트로더를 사용하여 플래시 프로그래밍을 수행할 경우, 앞서 STEP 3-1 에서 진행된 Configuration tool 설정 및 이로 인한 소스 코드 생성 이후, RTD > include 폴더내에 Ftfc_Fls_Ip.h 파일이, RTD > src 폴더내 Ftfc_Fls_Ip.c 파일이 각각 생성됩니다.

본 파일내 함수를 easyDSP 부트로더가 사용하는데, 이 함수가 램에서 동작하도록 설정해줘야 합니다.

Ftfc_Fls_Ip.h 파일내 중간 부분, 함수 선언이 시작되는 곳에 #define FLS_START_SEC_CODE 부분을 찾아 주석 처리한 후, 대신 #define FLS_START_SEC_RAMCODE 라인을 추가합니다.

그리고 파일 마지막 부분, 함수 선언이 끝나는 곳에서 #define FLS_STOP_SEC_CODE 부분을 찾아 주석 처리한 후, 대신 #define FLS_STOP_SEC_RAMCODE 라인을 추가합니다.

```
/*==================================================================================================
*                                        FUNCTION PROTOTYPES
==================================================================================================*/
//#define FLS_START_SEC_CODE    // commented
#define FLS_START_SEC_RAMCODE   // applied for code in ram
#include "Fls_MemMap.h"

/**
 * @brief        Initializes the FTCF module
 *
 * @details      This function will initialize ftfc module and clear all error flags.
 *
 * @param[in]    Ftfc_Fls_Ip_pInitConfig   Pointer to the driver configuration structure.
 * @return       Ftfc_Fls_Ip_StatusType
 * @retval       STATUS_FTFC_FLS_IP_SUCCESS            Initialization is success
 * @retval       STATUS_FTFC_FLS_IP_ERROR_TIMEOUT      Errors Timeout because wait for the Done bit long time
 *
 */
Ftfc_Fls_Ip_StatusType Ftfc_Fls_Ip_Init(const Ftfc_ConfigType * Ftfc_Fls_Ip_pInitConfig);

...
...
...
...
...
...

//#define FLS_STOP_SEC_CODE     // commented
#define FLS_STOP_SEC_RAMCODE    // applied for code in ram
#include "Fls_MemMap.h"


#ifdef __cplusplus
}
#endif

/** @} */

#endif /* FTFC_FLS_IP_H */
```

마찬가지로 Ftfc_Fls_Ip.c 파일내 중간 부분에서 각종 static 함수를 선언한 부분을 찾아 동일하게 처리합니다.

```
//#define FLS_START_SEC_CODE    // commented by easyDSP
#define FLS_START_SEC_RAMCODE   // added by easyDSP
#include "Fls_MemMap.h"

static boolean Ftfc_Fls_Ip_CheckValidRange(uint32 startAddress, uint32 length);
static Ftfc_Fls_Ip_StatusType Ftfc_Fls_Ip_ReadPreCheck(uint32 u32SrcAddress, const uint8 *pDestAddressPtr, uint32 u32Length);
static Ftfc_Fls_Ip_StatusType Ftfc_Fls_Ip_ComparePreCheck(uint32 u32SrcAddress, uint32 u32Length);
static Ftfc_Fls_Ip_StatusType Ftfc_Fls_Ip_SectorErasePreCheck(uint32 u32SectorStartAddress);
static Ftfc_Fls_Ip_StatusType Ftfc_Fls_Ip_WritePreCheck(uint32 u32DestAddress, const uint8 *pSourceAddressPtr, uint32 u32Length);

...
...
...

static Ftfc_Fls_Ip_StatusType Ftfc_Fls_Flash_AbortSuspended(void);
static void Ftfc_Fls_Ip_CalculateDFlashBitSize(void);

#if (STD_ON == FTFC_FLS_IP_SYNCRONIZE_CACHE)
static void Ftfc_Fls_SynchronizeCache(uint32 address, uint32 length);
#endif

//#define FLS_STOP_SEC_CODE     // commented by easyDSP
#define FLS_STOP_SEC_RAMCODE    // added by easyDSP
#include "Fls_MemMap.h"
```

또한 Ftfc_Fls_Ip.c 파일내 Ftfc_Fls_Ip_SectorErase 함수가 정의된 부분을 찾아, Ftfc_Fls_Ip_SectorErasePreCheck 함수를 호출하는 부분을 주석처리합니다.

```
Ftfc_Fls_Ip_StatusType Ftfc_Fls_Ip_SectorErase(uint32 u32SectorStartAddress)
{
    Ftfc_Fls_Ip_StatusType eRetVal;
    boolean bAddressValid  = FTFC_ADDRESS_VALID(u32SectorStartAddress);
    boolean bSectorAligned = FTFC_SECTOR_ALIGNED(u32SectorStartAddress);

    DEV_ASSERT_FTFC(bAddressValid);
    DEV_ASSERT_FTFC(bSectorAligned);
    /* Unused variables */
    (void)bAddressValid;
    (void)bSectorAligned;

    /* Check(if erase suspended is possible) if any ongoing erase suspended and abort it */
    eRetVal = Ftfc_Fls_Flash_AbortSuspended();

    if (STATUS_FTFC_FLS_IP_SUCCESS == eRetVal)
    {
        /* Pre-check before starting erase operation */
        //eRetVal = Ftfc_Fls_Ip_SectorErasePreCheck(u32SectorStartAddress); // commented by easyDSP
    }
    .
    .
    .
    .
```

Configuration Tool 이 자동 생성한 소스를 수정한 것이므로, Configuration tools 이 이를 감지하여, 소스파일을 다시 원래로 되돌릴 지 문의할 경우 이를 허용하지 마십시요.

STEP 4-2 : S32K3x 에서 easyDSP 부트로더를 위한 소스 코드 수정

S32K3x 에서 easyDSP 부트로더를 사용하여 플래시 프로그래밍을 수행할 경우, 앞서 STEP 3-2 에서 진행된 Configuration tool 설정 및 이로 인한 소스 코드 생성 이후, RTD > include 폴더내에 C40_Ip.h 파일이, RTD > src 폴더내 C40_Ip.c 파일이 각각 생성됩니다.

본 파일내 함수를 easyDSP 부트로더가 사용하는데, 이 함수가 램에서 동작하도록 설정해줘야 합니다.
C40_Ip.h 파일내 중간 부분에서 #define FLS_START_SEC_CODE 부분을 찾아 주석 처리한 후, 대신 #define FLS_START_SEC_RAMCODE 라인을 추가합니다.
그리고 파일 마지막 부분에서 #define FLS_STOP_SEC_CODE 부분을 찾아 주석 처리한 후, 대신 #define FLS_STOP_SEC_RAMCODE 라인을 추가합니다.

```
/*==================================================================================================
*                                        FUNCTION PROTOTYPES
==================================================================================================*/
//#define FLS_START_SEC_CODE    // commented by easyDSP
#define FLS_START_SEC_RAMCODE   // added by easyDSP
#include "Fls_MemMap.h"

/**
 * @brief        Initializes the C40 module
 *
 * @details      This function will initialize c40 module and clear all error flags.
 *
 * @param[in]    InitConfig   Pointer to the driver configuration structure.
 * @return       C40_Ip_StatusType
 * @retval       STATUS_C40_IP_SUCCESS            Initialization is success
 * @retval       STATUS_C40_IP_ERROR_TIMEOUT      Errors Timeout because wait for the Done bit long time
 *
 */
C40_Ip_StatusType C40_Ip_Init(const C40_ConfigType * InitConfig);

...
...
...
...


/**
 * @brief    Set synch/Asynch at IP layer base on the bAsynch of HLD
 *
 * @details  This function will change C40_Ip_Async value at IP layer. Its param base on the bAsynch of HLD.
 *           Thanks for this param, writting and erasing will operate at synch or Asynch mode.
 *
 * @pre     The module has to be initialized
 *
 */
void C40_Ip_SetAsyncMode(const boolean Async);

//#define FLS_STOP_SEC_CODE     // commented by easyDSP
#define FLS_STOP_SEC_RAMCODE    // added by easyDSP
#include "Fls_MemMap.h"
```

마찬가지로 C40_Ip.c 파일내 중간 부분에서 각종 static 함수를 선언한 부분을 찾아 동일하게 처리합니다.

```
/*==================================================================================================
                                        LOCAL FUNCTION PROTOTYPES
==================================================================================================*/
//#define FLS_START_SEC_CODE    // commented by easyDSP
#define FLS_START_SEC_RAMCODE   // added by easyDSP
#include "Fls_MemMap.h"

static inline uint32 C40_Ip_ReadData32(uint32 Address);
static inline uint16 C40_Ip_ReadData16(uint32 Address);
static inline uint8 C40_Ip_ReadData8(uint32 Address);

...
...
...

static inline boolean C40_Ip_ValidUTestAddress(uint32 Address);
static inline boolean C40_Ip_ValidAddress(uint32 Address);
static inline boolean C40_Ip_ValidRangeAddress(uint32 StartAddress,
                                               uint32 Length
                                              );

//#define FLS_STOP_SEC_CODE     // commented by easyDSP
#define FLS_STOP_SEC_RAMCODE    // added by easyDSP
#include "Fls_MemMap.h"
```

Configuration Tool 이 자동 생성한 소스를 수정한 것이므로, Configuration tools 이 이를 감지하여, 소스파일을 다시 원래로 되돌릴 지 문의할 경우 이를 허용하지 마십시요.

STEP 4-3 : S32M24x 에서 easyDSP 부트로더를 위한 소스 코드 수정

S32M24x 에서 easyDSP 부트로더를 사용하여 플래시 프로그래밍을 수행할 경우, 앞서 STEP 3-3 에서 진행된 Configuration tool 설정 및 이로 인한 소스 코드 생성 이후, RTD > include 폴더내에 Ftfc_Mem_InFls_Ip.h 파일이, RTD > src 폴더내 Ftfc_Mem_InFls_Ip.c 파일이 각각 생성됩니다.

본 파일내 함수를 easyDSP 부트로더가 사용하는데, 이 함수가 램에서 동작하도록 설정해줘야 합니다.
Ftfc_Mem_InFls_Ip.h 파일내 중간 부분, 함수 선언이 시작되는 곳에 #define MEM_43_INFLS_START_SEC_CODE 부분을 찾아 주석 처리한 후, 대신 #define MEM_43_INFLS_START_SEC_RAMCODE 라인을 추가합니다.
그리고 파일 마지막 부분, 함수 선언이 끝나는 곳에서 #define MEM_43_INFLS_STOP_SEC_CODE 부분을 찾아 주석 처리한 후, 대신 #define MEM_43_INFLS_STOP_SEC_RAMCODE 라인을 추가합니다.

```
/*==================================================================================================
*                                       FUNCTION PROTOTYPES
==================================================================================================*/
//#define MEM_43_INFLS_START_SEC_CODE   // commented by easyDSP
#define MEM_43_INFLS_START_SEC_RAMCODE  // added by easyDSP
#include "Mem_43_INFLS_MemMap.h"

/**
 * @brief        Initializes the FTCF module
 *
 * @details      This function will initialize ftfc module and clear all error flags.
 *
 * @param[in]    Ftfc_Mem_InFls_Ip_pInitConfig   Pointer to the driver configuration structure.
 * @return       Ftfc_Mem_InFls_Ip_StatusType
 * @retval       STATUS_FTFC_MEM_INFLS_IP_SUCCESS            Initialization is success
 * @retval       STATUS_FTFC_MEM_INFLS_IP_ERROR_TIMEOUT      Errors Timeout because wait for the Done bit long time
 *
 */
Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_Init(const Ftfc_Mem_InFls_Ip_ConfigType * Ftfc_Mem_InFls_Ip_pInitConfig);

...
...
...

Ftfc_Mem_InFls_Ip_StatusType Ftfc_Mem_InFls_Ip_CtrlStatus(void);

void Ftfc_Mem_InFls_Ip_ReportEccUnCorrectedError(void);

/**
 * @brief       Set Status for Ftfc_Mem_InFls_Ip_LoadAc_Status
 *
 * @param[in]   Status
 */
void Ftfc_Mem_InFls_Ip_SetLoadAcStatus(const boolean Status);

//#define MEM_43_INFLS_STOP_SEC_CODE    // commented by easyDSP
#define MEM_43_INFLS_STOP_SEC_RAMCODE   // added by easyDSP
#include "Mem_43_INFLS_MemMap.h"
```

마찬가지로 Ftfc_Mem_InFls_Ip.c 파일내 중간 부분에서 각종 static 함수를 선언한 부분을 찾아 앞과 동일하게 처리합니다.

```
//#define MEM_43_INFLS_START_SEC_CODE    // commented by easyDSP
#define MEM_43_INFLS_START_SEC_RAMCODE  // added by easyDSP
#include "Mem_43_INFLS_MemMap.h"

static void Ftfc_Mem_InFls_Ip_FlashAccessCalloutStart(void);
static void Ftfc_Mem_InFls_Ip_FlashAccessCalloutFinish(void);

static void Ftfc_Mem_InFls_Ip_InvalidPrefetchBuff(void);
static void Ftfc_Mem_InFls_Ip_CalculateDFlashBitSize(void);

...
...
...

static void Ftfc_Mem_InFls_Ip_ProgramVerify(uint32 address, const uint8 * data, uint32 size);
static void Ftfc_Mem_InFls_Ip_EraseVerify(uint32 address, uint32 size);

static void Ftfc_Mem_InFls_Ip_ClearErrorFlags(void);
static void Ftfc_Mem_InFls_Ip_LoadFCCOBParams(const uint32 u32PhysicAddr,
                                    const uint8 *pDataAddr,
                                    const uint8  u8FCCOBCmdId
                                   );

#if (STD_ON == FTFC_MEM_INFLS_IP_SYNCHRONIZE_CACHE)
static void Ftfc_Mem_InFls_Ip_SynchronizeCache(uint32 address, uint32 length);
#endif

//#define MEM_43_INFLS_STOP_SEC_CODE     // commented by easyDSP
#define MEM_43_INFLS_STOP_SEC_RAMCODE   // added by easyDSP
#include "Mem_43_INFLS_MemMap.h"
```

STEP 4-4 : S32M27x 에서 easyDSP 부트로더를 위한 소스 코드 수정

S32M27x 에서 easyDSP 부트로더를 사용하여 플래시 프로그래밍을 수행할 경우, 앞서 STEP 3-4 에서 진행된 Configuration tool 설정 및 이로 인한 소스 코드 생성 이후, RTD > include 폴더내에 C40_Ip.h 파일이, RTD > src 폴더내 C40_Ip.c 파일이 각각 생성됩니다.

본 파일내 함수를 easyDSP 부트로더가 사용하는데, 이 함수가 램에서 동작하도록 설정해줘야 합니다.
C40_Ip.h 파일내 중간 부분, 함수 선언이 시작되는 곳에 #define MEM_43_INFLS_START_SEC_CODE 부분을 찾아 주석 처리한 후, 대신 #define MEM_43_INFLS_START_SEC_ RAMCODE 라인을 추가합니다.
그리고 파일 마지막 부분, 함수 선언이 끝나는 곳에서 #define MEM_43_INFLS_STOP_SEC_CODE 부분을 찾아 주석 처리한 후, 대신 #define MEM_43_INFLS_STOP_SEC_RAMCODE 라인을 추가합니다.

```
/*=================================================================================================
*                                       FUNCTION PROTOTYPES
=================================================================================================*/
//#define MEM_43_INFLS_START_SEC_CODE   // commented by easyDSP
#define MEM_43_INFLS_START_SEC_RAMCODE  // added by easyDSP
#include "Mem_43_INFLS_MemMap.h"

#if (STD_ON == C40_IP_MAIN_INTERFACE_ENABLED)
/**
 * @brief    Set synch/Asynch at IP layer base on the bAsynch of HLD
 *
 * @details  This function will change C55_Ip_bAsync value at IP layer. Its param base on the bAsynch of HLD.
 *           Thanks for this param, writing and erasing will operate at synch or Asynch mode.
 *
 * @pre     The module has to be initialized
 *
 */
void C40_Ip_SetAsyncMode(const boolean Async);
#endif

...
...
...
...

/**
 * @brief    Get the failing address in memory
 *
 * @details  This function will get the failing address in the event of ECC
 *           event error, Single Bit Correction, as well as providing the address of a
 *           failure that may have occurred in a program/erase operation.
 *
 * @return  uint32
 * @retval  Return the address is failed in the event or single bit correction.
 *
 * @pre     The module has to be initialized
 *
 */
uint32 C40_Ip_GetFailedAddress(void);

//#define MEM_43_INFLS_STOP_SEC_CODE    // commented by easyDSP
#define MEM_43_INFLS_STOP_SEC_RAMCODE   // added by easyDSP
#include "Mem_43_INFLS_MemMap.h"
```

마찬가지로 C40_Ip. c 파일내 중간 부분에서 각종 static 함수를 선언한 부분을 찾아 앞과 동일하게 처리합니다.

```
/*=================================================================================================
                                       LOCAL FUNCTION PROTOTYPES
=================================================================================================*/
//#define MEM_43_INFLS_START_SEC_CODE   // commented by easyDSP
#define MEM_43_INFLS_START_SEC_RAMCODE  // added by easyDSP
#include "Mem_43_INFLS_MemMap.h"

/* Trusted Function */
#if (STD_ON == C40_IP_ENABLE_USER_MODE_SUPPORT)
void C40_Ip_SetUserAccessAllowed(void);
#endif

    ...
    ...
    ...
    ...

#if ((C40_IP_ECC_CHECK == STD_ON) || (C40_IP_ECC_CHECK_BY_AUTOSAR_OS == STD_ON))
void C40_Ip_ReportEccUnCorrectedError(void)
{
    /* save read status */
    C40_Ip_eReadStatus = C40_IP_STATUS_ECC_UNCORRECTED;
}
#endif

//#define MEM_43_INFLS_STOP_SEC_CODE    // commented by easyDSP
#define MEM_43_INFLS_STOP_SEC_RAMCODE   // added by easyDSP
#include "Mem_43_INFLS_MemMap.h"

#ifdef __cplusplus
}
#endif
```

STEP 5 : easyDSP 제공 소스 파일 및 함수 호출

easyDSP 통신 및 플래시프로그래밍을 위해 제공되는 소스파일 (easyS32_RTD.h, easyS32_RTD_comm.c, easyS32_RTD_boot.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더 아래 ₩source₩S32 에서 찾을 수 있습니다.

이중 easyS32_RTD.h 파일 일부 내용을 수정해야 합니다.우선 easyDSP 통신을 위해 선정된 LPUART 채널을 지정합니다. 아래 예제에서는 LPUART0 으로 선정하였습니다.
만약 easyDSP 를 사용하여 플래시 프로그래밍을 수행할 경우, EZ_BOOTLOADER_USE 를 1 로 설정하시기 바랍니다.

```
/*****************************************************************************
 Select UART channel for easyDSP communication
*****************************************************************************/
#define EZ_UART_CH      0           // LPUART0 for easyDSP

/*****************************************************************************
 In case you use the boot loader provided by easyDSP to program flash,
 define EZ_BOOTLOADER_USE as 1. Otherwise, define as 0.
*****************************************************************************/
#define EZ_BOOTLOADER_USE 1
```

main.c 상단에 easyS32_RTD.h 를 include 하여 주시고, main 함수에서 초기화 루틴 이후 easyDSP_init() 함수를 호출하시기 바랍니다. easyDSP_init() 함수에서는 easyDSP 통신을 위한 각종 설정을 수행합니다.
클럭, 핀, 인터럽트 설정은 easyDSP 통신을 위해 필요하므로 main 함수 앞부분에서 수행되어야 합니다.
만약 easyDSP 가 플래시 프로그래밍을 지원하길 원하시면 클럭 및 핀이 설정된 바로 직후 easyDSP_boot()를

호출하시기 바랍니다.

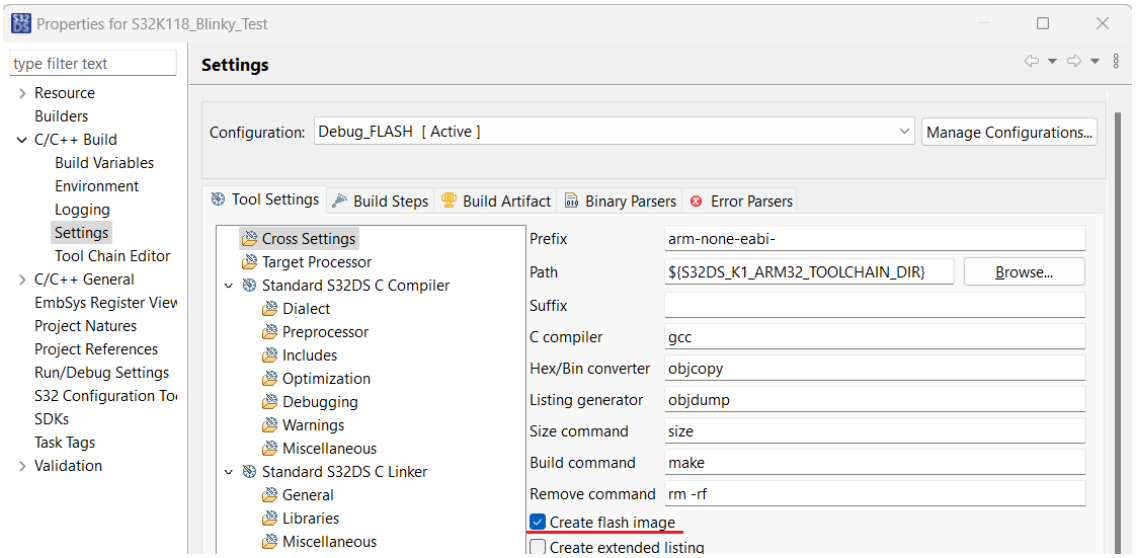
```
#include "easyS32_RTD.h"

int main(void)
{
    // Init clock
    Clock_Ip_Init(&Clock_Ip_aClockConfig[0]);
#if defined (FEATURE_CLOCK_IP_HAS_SPLL_CLK)
    // Busy wait until the System PLL is locked
    while (CLOCK_IP_PLL_LOCKED != Clock_Ip_GetPllStatus());
    Clock_Ip_DistributePll();
#endif

    // Initialize all pins in case of S32K1
    Port_Ci_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);

    // Initialize all pins in case of S32K3
    Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);

#if EZ_BOOTLOADER_USE
    // Right after clock and pin setting, call easyDSP_boot() to enable flash programming
    easyDSP_boot();
#endif

    // Initialize IRQs
    IntCtrl_Ip_Init(&IntCtrlConfig_0);
    IntCtrl_Ip_ConfigIrqRouting(&intRouteConfig);

    // reset of initial setting
    .
    .
    .
    .
    .
    .

    // call easyDSP_init() to enable easyDSP monitoring
    easyDSP_init();

    // loop forever
    while(1)
    {
        .
        .
        .
    }
}
```

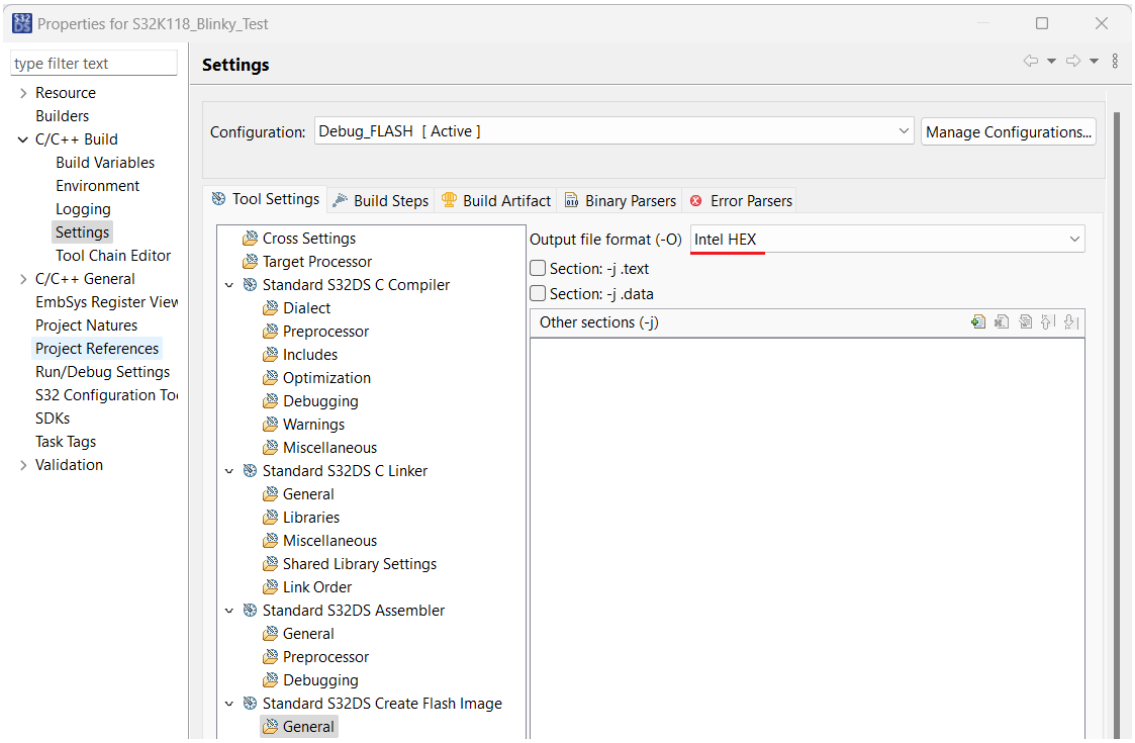
STEP 6 : IDE 설정

1. 매 컴파일마다 hex 파일 (인텔 형식) 이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 IDE 를 설정해주세요. Hex 파일은 플래시 프로그래밍할 때 사용됩니다.

Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.

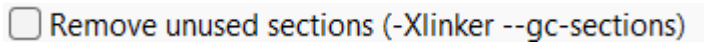
S32DS 기준 하기 옵션 설정 참조하세요.





2. easyDSP 로 모니터링을 수행하기 위해서는, 출력 파일 (예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

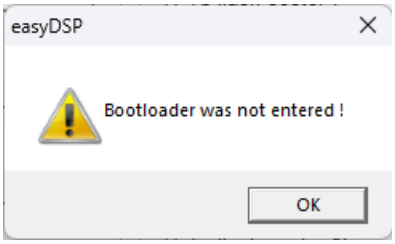
3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 해당 변수를 모니터링하려면 적절한 IDE 세팅이 필요합니다. 예를 들어, S32DS 의 프로젝트 설정 > 링커 옵션 하기 항목을 체크하지 않습니다.



STEP 7 : easyDSP 부트로더의 제한 사항

1. S32K MCU 는 제작업체에서 제공하는 롬부트로더가 없기에 사용자가 부트로더를 별도로 제작 사용해야 합니다. easyDSP 는 사용자 프로그램 안에 easyDSP 가 제공하는 부트로더 (함수 easyDSP_boot)가 존재하는 구성입니다. 따라서 MCU 플래시에 easyDSP 소스파일이 이미 프로그래밍되어 있어야 새로운 프로그램으로 재

프로그래밍이 가능합니다. 만약 플래시가 전부 지워져 있거나, easyDSP 소스파일이 플래시에 프로그래밍이 되어 있지 않다면, 플래시 프로그래밍이 지원되지 않으며 하기와 같은 메시지가 송출되며, 이 경우 디버거를 사용한 플래시 프로그래밍이 필요합니다. 따라서 처음 한번은 디버거를 이용해서 플래시 프로그래밍을 해야만 합니다.



2. easyDSP 부트로더는 램에서 동작해야 하기에 일정 램 영역을 소비합니다.
S32K1 의 경우 약 2.4K 바이트, S32K3 의 경우 약 4.8kB 입니다 (-O1 최적화 옵션 기준).

7.4 AM263x

7.4.1 AM263x 소프트웨어 설정

STEP 1 : 코어 선정

easyDSP 측면에서 코어는 총 4 가지 종류로 구분됩니다.
노란색 코어 : easyDSP Pod 가 연결되고 easyDSP 모니터링 수행되는 코어
주황색 코어 : easyDSP Pod 가 연결되지는 않지만 easyDSP 모니터링이 수행되는 코어
파란색 코어 : easyDSP 모니터링이 수행되지 않는 코어
회색 코어 : 동작하지 않는 코어입니다.

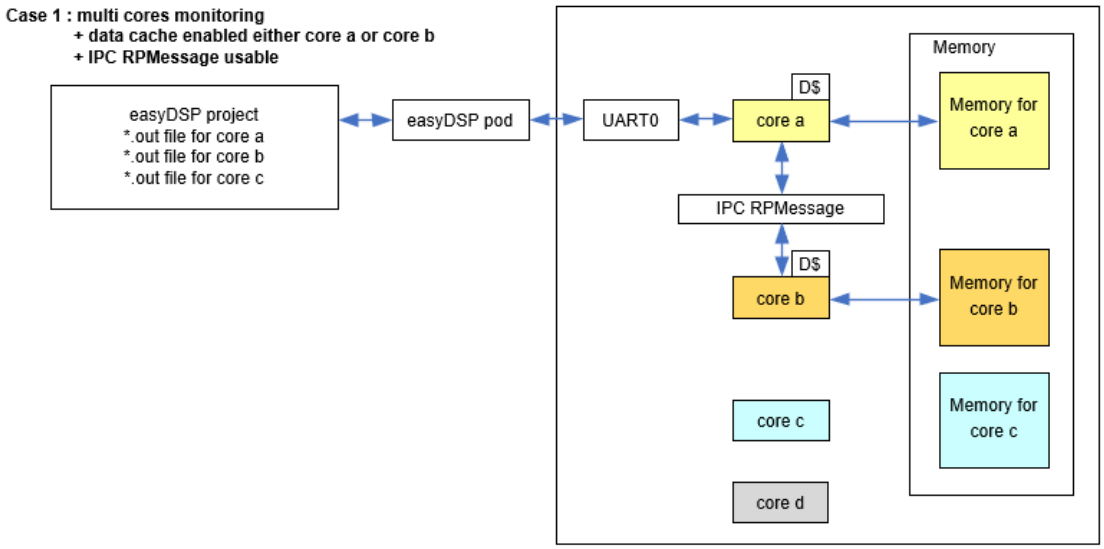
	easyDSP 포드에 연결	easyDSP와 통신 여부	코어 동작 여부
코어	**연결**	**통신**	**동작**
코어	미연결	**통신**	**동작**
코어	미연결	미통신	**동작**
코어	미연결	미통신	미동작

AM263x 시리즈는 최대 4 개의 코어가 제공되므로, 사용자 시스템에 맞춰 해당 코어를 선정하세요.
파란색 코어와 회색 코어는 easyDSP 와 관련 없으므로 easyDSP 관련 작업을 수행하지 않습니다.
따라서 노란색 코어, 주황색 코어를 선정해야 합니다. AM263x 의 어떤 코어든 선정 가능합니다.
또한 데이타 캐시 사용 여부 등에 따라 아래와 같이 여러 케이스로 구성할 수 있습니다.

Case 1 :

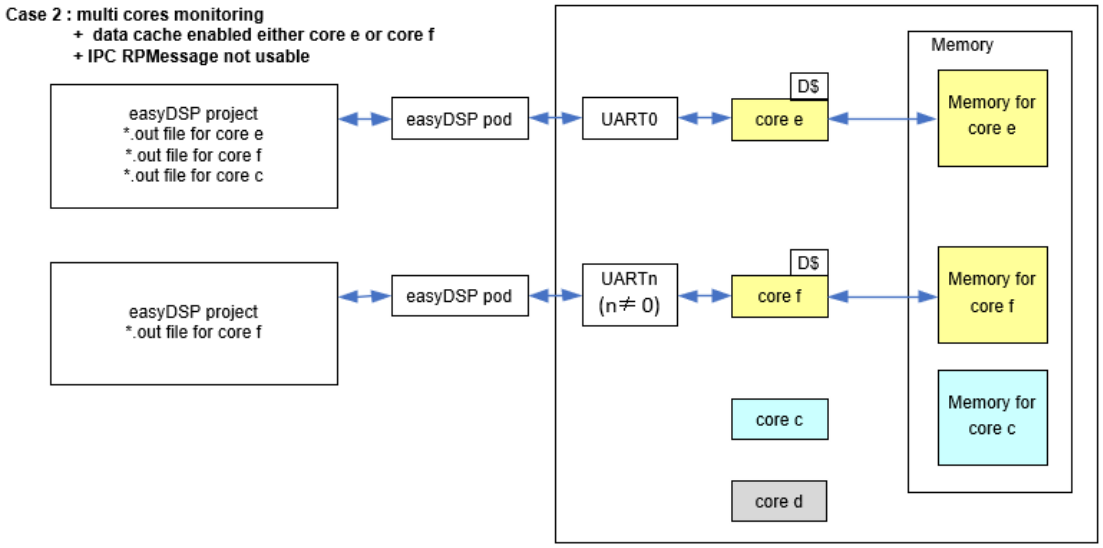
본 경우는 easyDSP 가 여러 코어 (core a, b)에 대해 모니터링을 수행하며, 그 중 데이타 캐시를 사용하는 코어가 하나라도 있고, IPC RPMessage 를 사용할 수 있는 경우입니다.
easyDSP pod 는 UART0 을 거쳐 core a 에 연결되며, core a 변수가 저장된 메모리는 core a 가 억세스합니다.

그리고 캐시 일관성 문제를 회피하기 위해, core b 변수가 저장된 메모리는 core b 가 억세스합니다. 이를 구현하기 위한 코어간 통신은 IPC RPMessage 를 사용합니다.
easyDSP 모니터링을 위해 데이타가 이동하는 경로는 화살표를 참조하세요.



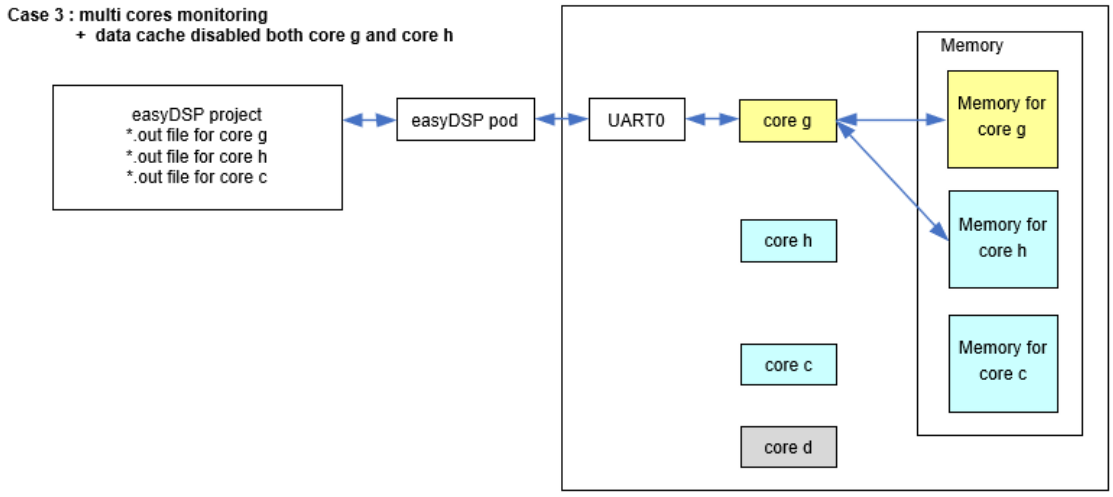
Case 2 :

Case 1 경우이지만 코어간 통신을 위해 IPC RPMessage 를 사용할 수 없는 경우입니다.
각각의 코어에 easyDSP pod 를 연결해야 하며 이 경우 하기와 같은 구성이 됩니다.



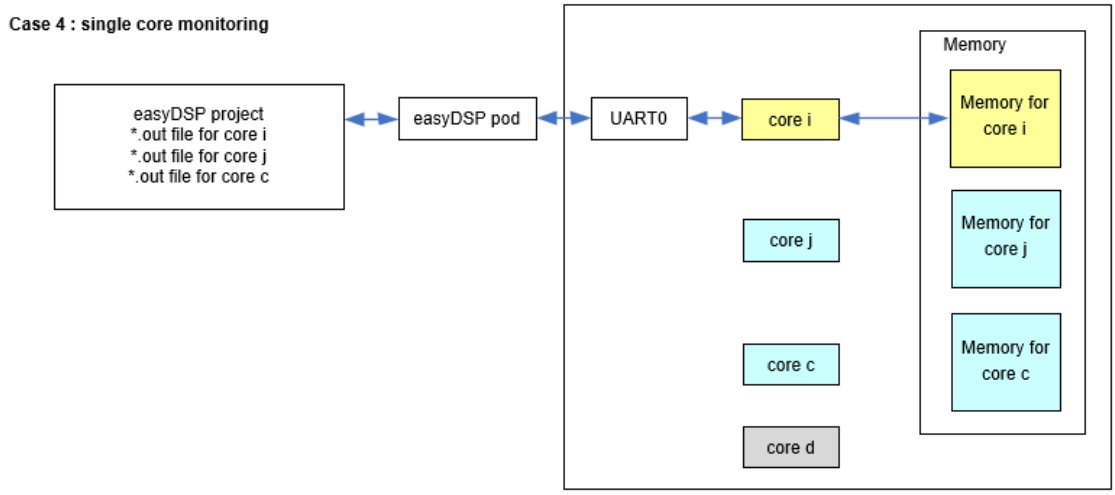
Case 3 :

easyDSP 가 여러 코어에 대해 모니터링을 수행하지만, 그 중 데이타 캐시를 사용하는 코어가 하나도 없는 경우입니다.
이 경우 easyDSP pod 이 연결된 코어에서 직접 해당 메모리를 접근하여 변수 모니터링을 수행합니다.



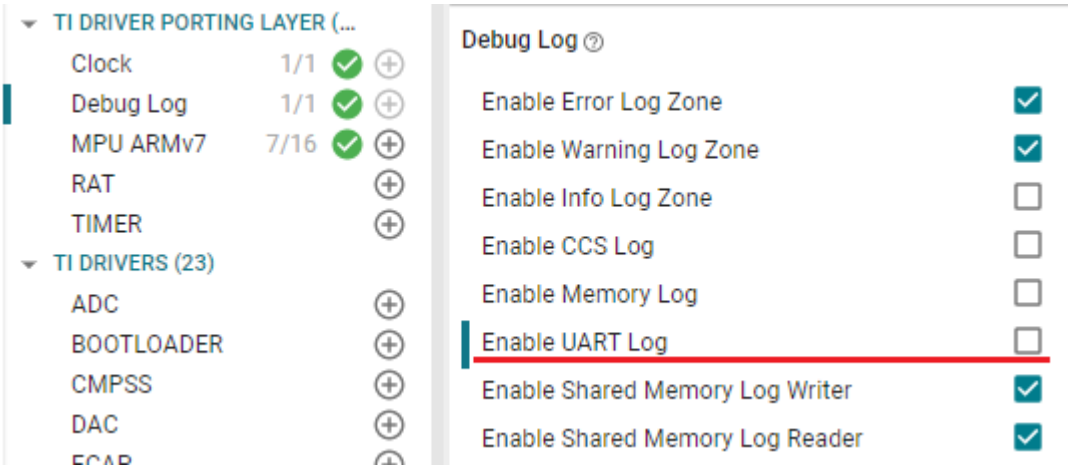
Case 4 :

easyDSP 가 하나의 코어에 대해서만 모니터링을 수행하는 경우입니다. 해당 코어의 데이타 캐시 사용 여부와 상관없습니다.

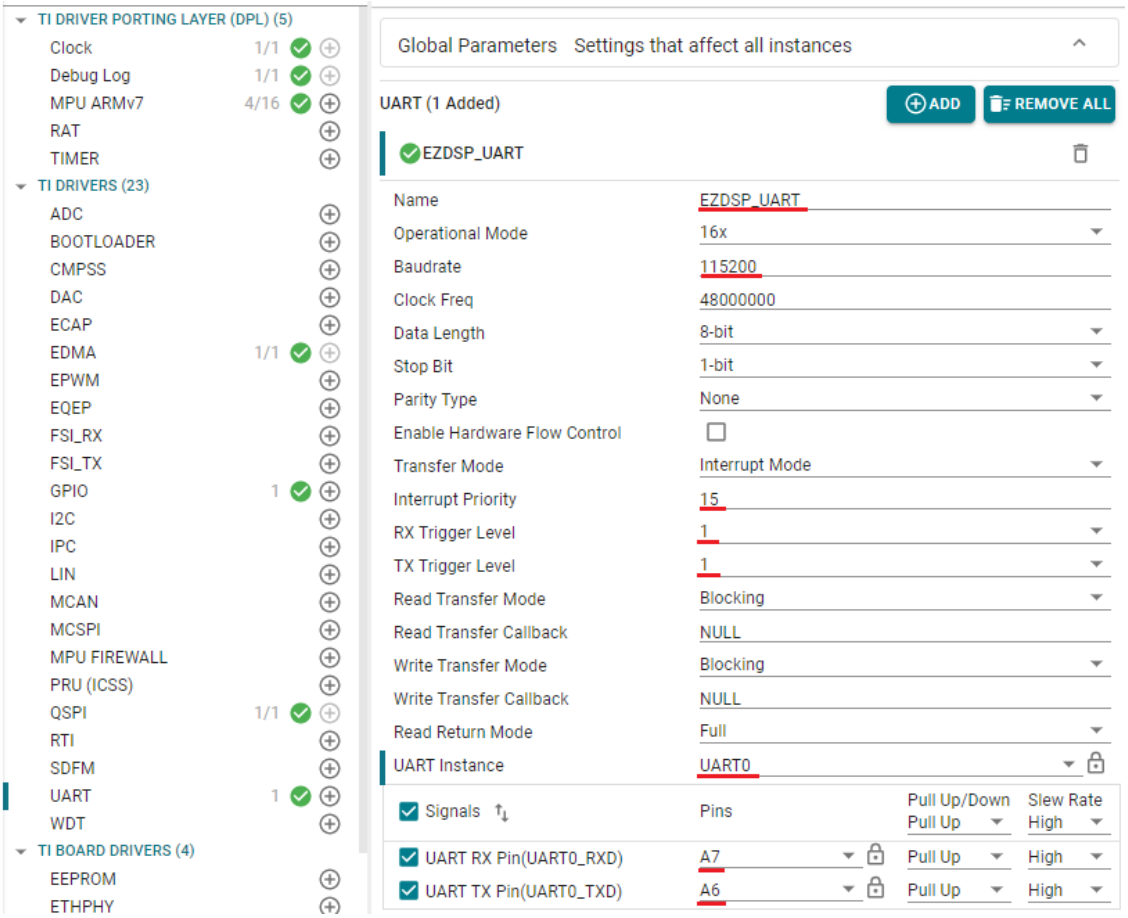


STEP 2 : SysConfig 설정

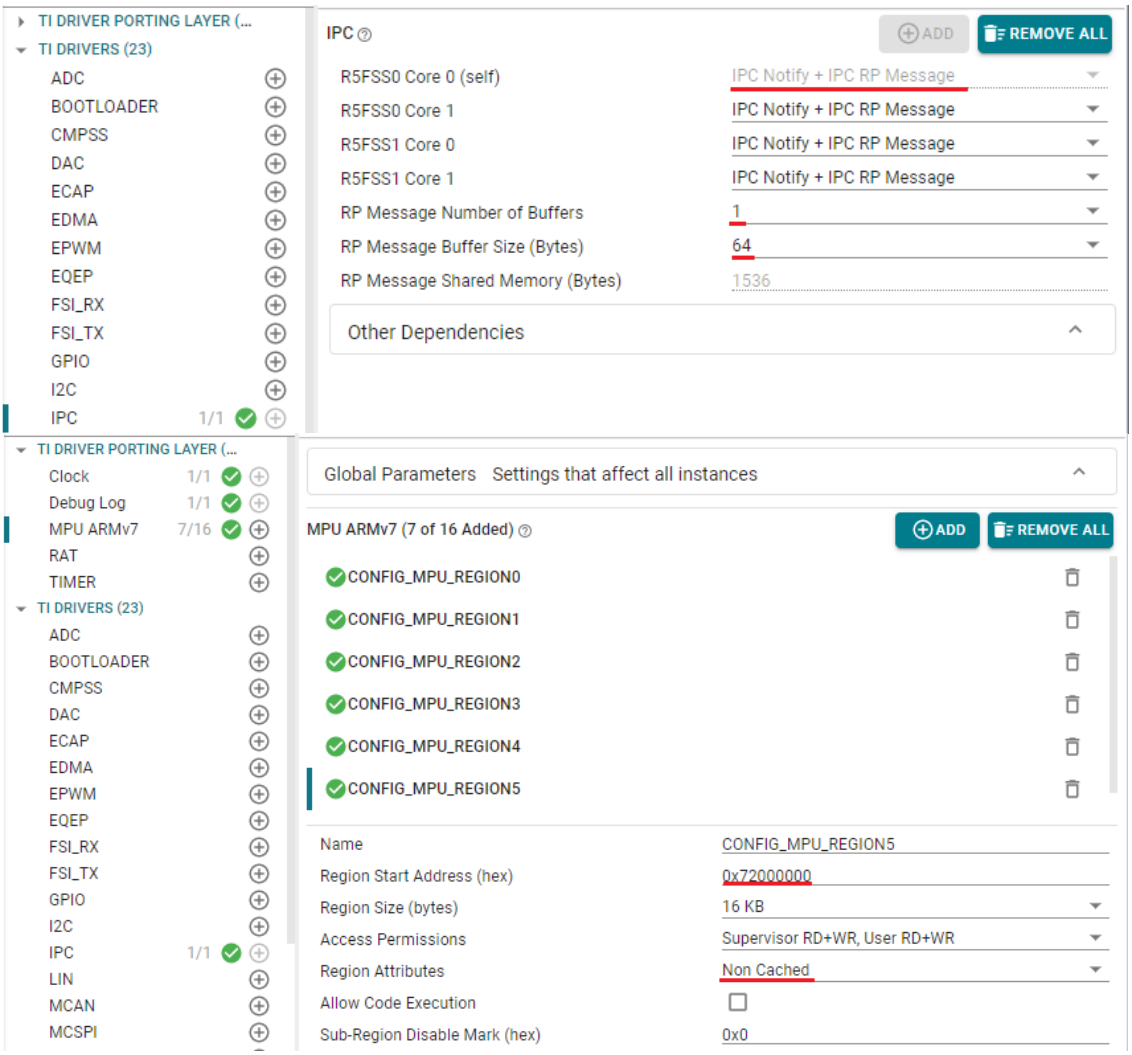
easyDSP 는 SysConfig 기반 생성된 코드를 사용합니다. 하기에서는 SysConfig 1.13.0 기준으로 설명드립니다.
easyDSP 는 MCU 와 통신시 UART0 을 사용하므로, Debug Log > Enable UART Log 를 비활성화거나, 사용이 필요할 경우 UART0 이 아닌 다른 곳으로 설정합니다.



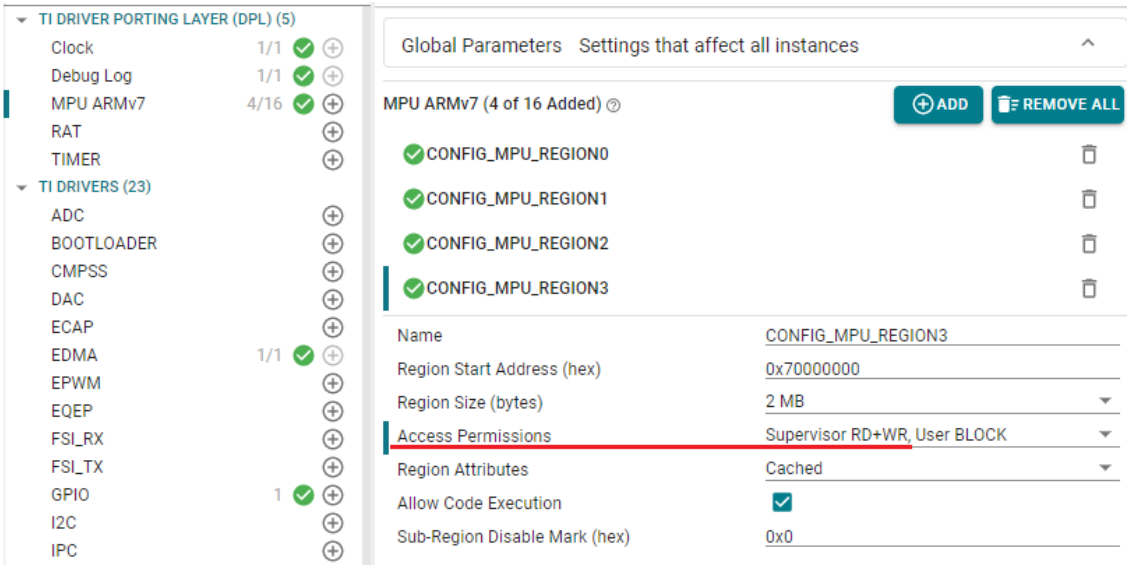
easyDSP Pod 가 연결된 코어 (STEP1 의 노란색 코어)에 대해서 UART 관련 설정을 하기와 같이 수행합니다.
먼저 이름은 반드시 EZDSP_UART 이어야 하며, Baudrate 는 사용자가 적절히 선정 가능하지만 easyDSP 프로젝트 설정값과 동일해야 합니다.
8 비트 데이타, 하나의 stop bit, no parity bit 의 데이타 규격을 사용하며, 인터럽트 순위는 최하순위(값 15)로 선정합니다.
UART0 을 사용하셔야 하며, MUXMODE 0 인 TX, RX 핀을 할당해야 합니다. 예외적으로 STEP1 의 core f 는 UART0 이 아닌 다른 UART 를 사용합니다.
기타 부분은 하기 그림 참조 바랍니다.



IPC RPMessage 를 사용하는 코어(core a, b)에 대해서 IPC 설정이 필요합니다.
통신 방식은 'IPC Notify + IPC RP Message', 버퍼 개수는 최소 1, 버퍼 사이즈는 최소 64 가 필요합니다.
사용자가 별도의 목적으로 IPC RPMessage 를 사용할 경우, 이 값 이상으로 선정할 수 있습니다.
또한 IPC RPMessage 사용시 공유 버퍼로 사용되는 메모리 영역 (0x72000000) 의 Non Cached 설정이 필요합니다.



easyDSP 가 모니터링을 수행하는 모든 코어에서 (core a, b, e, f, g, h, i) 변수가 위치한 메모리 영역을 easyDSP 가 읽고 쓸 수 있도록 해당 메모리 영역을 Supervion RD+WR 으로 설정합니다 (User 설정은 상관없음).



STEP 3 : easyDSP 프로젝트 및 사용자 프로젝트

STEP1 의 경우에 맞게, 노란색 코어에 대해 easyDSP 프로젝트를 생성하며, 노란색 및 주황색 코어에 대해 사용자 MCU 프로젝트를 easyDSP 모니터링이 가능하도록 수정해야 합니다.

노란색 및 주황색 코어에 대해, easyDSP 통신용 파일 (easyAM_v*.*.h, easyAM_v*.*.c)을 사용자 프로젝트에 포함하시기 바랍니다. 버전에 따라 파일명이 달라질 수 있습니다. 해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩AM2x 에서 찾을 수 있습니다.
그리고 아래 헤더 파일을 사용 환경에 맞춰 설정하시기 바랍니다.

```
////////////////////////////////////////////////////////////////////////////////////////////////////
// Specify whether easyDSP pod is connected to this core
// Define 1 if easyDSP pod is connected to this core
// Define 0 if easyDSP pod is not connected to this core
////////////////////////////////////////////////////////////////////////////////////////////////////
#define EASYDSP_POD_IS_CONNECTED_TO_THIS_CORE       1

////////////////////////////////////////////////////////////////////////////////////////////////////
// Specify whether easyDSP communicates with single core or multi cores
// Define 1 if easyDSP communicates with multi cores
// Define 0 if easyDSP communicates with single core
////////////////////////////////////////////////////////////////////////////////////////////////////
#define EASYDSP_IS_COMMUNICATING_WITH_MULTI_CORES     1

////////////////////////////////////////////////////////////////////////////////////////////////////
// If easyDSP communicates with multi cores, Specify data cache is enabled or not in that cores
// Define 1 if data cache is enabled in the at least one core easyDSP communicates with
// Define 0 if data cache is disabled in all the cores that easyDSP communicates with
////////////////////////////////////////////////////////////////////////////////////////////////////
#if EASYDSP_IS_COMMUNICATING_WITH_MULTI_CORES
#define D_CACHE_IS_ENABLED     1
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////
// If easyDSP communicates with multi cores with data cache enabled, Specify IPC RPMessage end point
// It should range from 0 to 63
////////////////////////////////////////////////////////////////////////////////////////////////////
#if EASYDSP_IS_COMMUNICATING_WITH_MULTI_CORES
#if D_CACHE_IS_ENABLED
#define MAIN_CORE_SERVICE_END_PT     (12U)
#define REMOTE_CORE_SERVICE_END_PT   (13U)
#endif
#endif
```

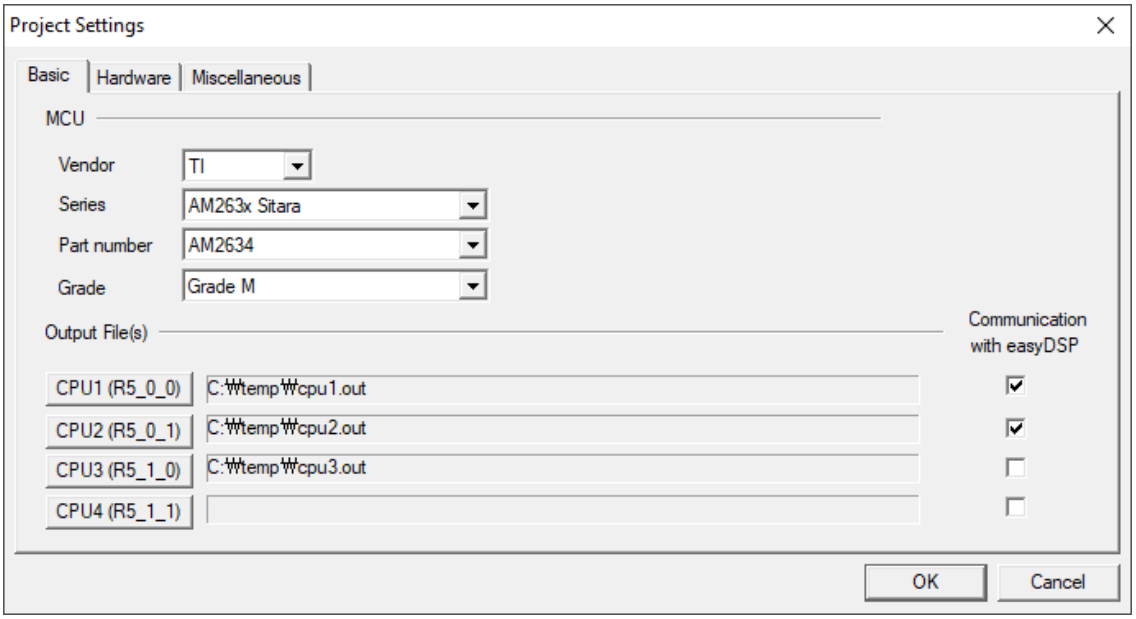
그리고 사용자 프로그램에서 각종 초기화 함수가 호출된 이후, easyDSP_init()을 호출하므로써, easyDSP 모니터링이 가능하게 합니다.

```
#include "easyAM_v*.*.h"
int main()
{
    System_init();
    Board_init();
    Drivers_open();
    Board_driversOpen();

    .
    .
    .
    .

    easyDSP_init();
    .
    .

}
```

하기에서 STEP 1 의 케이스별로 상세 설명합니다.

Case 1 :

만약 core a, b, c, d 가 각각 CPU1, 2, 3, 4 일 경우 하기처럼 프로젝트를 설정합니다.
easyDSP 와 통신하는 코어로 CPU1, CPU2 를 선정하며, 실제 동작하는 모든 코어의 출력 파일을 등록합니다.

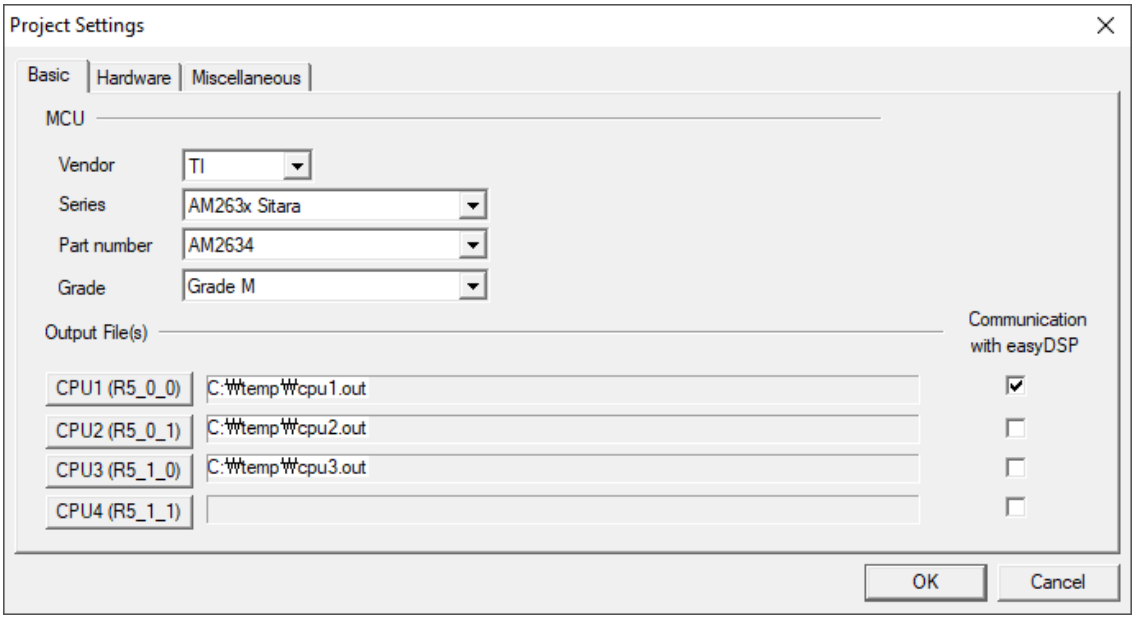


헤더 파일은 하기와 같이 설정합니다. 코어간 통신에 IPC RPMessage 를 사용하며 2 개의 엔드포인트 값(m, n)을 설정해야 합니다.
easyDSP pod 연결 여부만 차이가 나며, 나머지 변수는 동일한 값입니다.

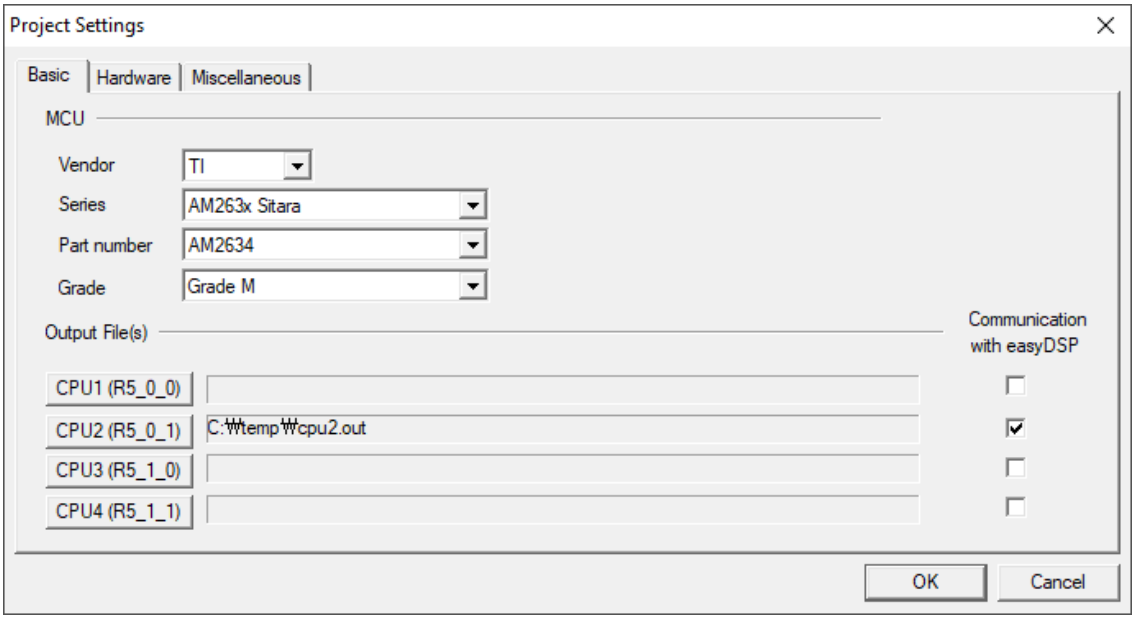
사용자 프로젝트	노란색 코어	주황색 코어
easyAM.h 설정	EASYDSP_POD_IS_CONNECTED_TO_THIS_CORE = **1** EASYDSP_IS_COMMUNICATING_WITH_MULTI_CORES =**1** D_CACHE_IS_ENABLED = **1** MAIN_CORE_SERVICE_END_PT = m REMOTE_CORE_SERVICE_END_PT = n	EASYDSP_POD_IS_CONNECTED_TO_THIS_CORE = **0** EASYDSP_IS_COMMUNICATING_WITH_MULTI_CORES =**1** D_CACHE_IS_ENABLED = **1** MAIN_CORE_SERVICE_END_PT = m REMOTE_CORE_SERVICE_END_PT = n

Case 2 :

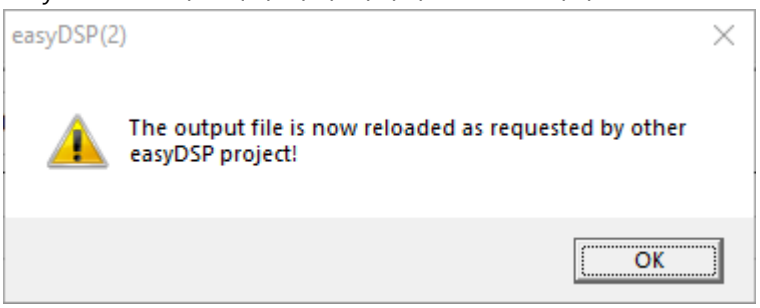
만약 core e, f, c, d 가 각각 CPU1, 2, 3, 4 일 경우, core e 용 easyDSP 프로젝트는 하기처럼 설정합니다.



core f 용 easyDSP 프로젝트 설정은 하기와 같습니다.



램부팅 및 플래시 프로그래밍을 수행하려면 easyDSP Pod 가 연결될 코어는 반드시 UART0 를 거쳐 연결되어야 합니다.
따라서 UART0 이 연결된 코어 core e 의 easyDSP 프로젝트에 동작하는 모든 코어의 출력파일을 등록하며 램부팅 및 플래시 프로그래밍을 수행합니다.
UART0 이 연결되지 않는 코어 core f 의 easyDSP 프로젝트에서는 모니터링만 가능하므로 램부팅 및 플래시 프로그래밍을 수행하지 마세요.
코어 f 의 사용자 프로그램이 변경되고 이를 코어 e 의 easyDSP 프로젝트에서 램부팅 또는 플래시 프로그래밍하게 되면, 코어 f 의 easyDSP 프로젝트는 출력 파일을 다시 읽어서 변수 정보를 갱신하게 됩니다.
이는 코어 e 와 코어 f 의 easyDSP 프로젝트가 하나의 PC 에서 수행될 때 자동적으로 수행되며 이후 코어 f 의 easyDSP 프로젝트에서 하기 메시지를 송출합니다.



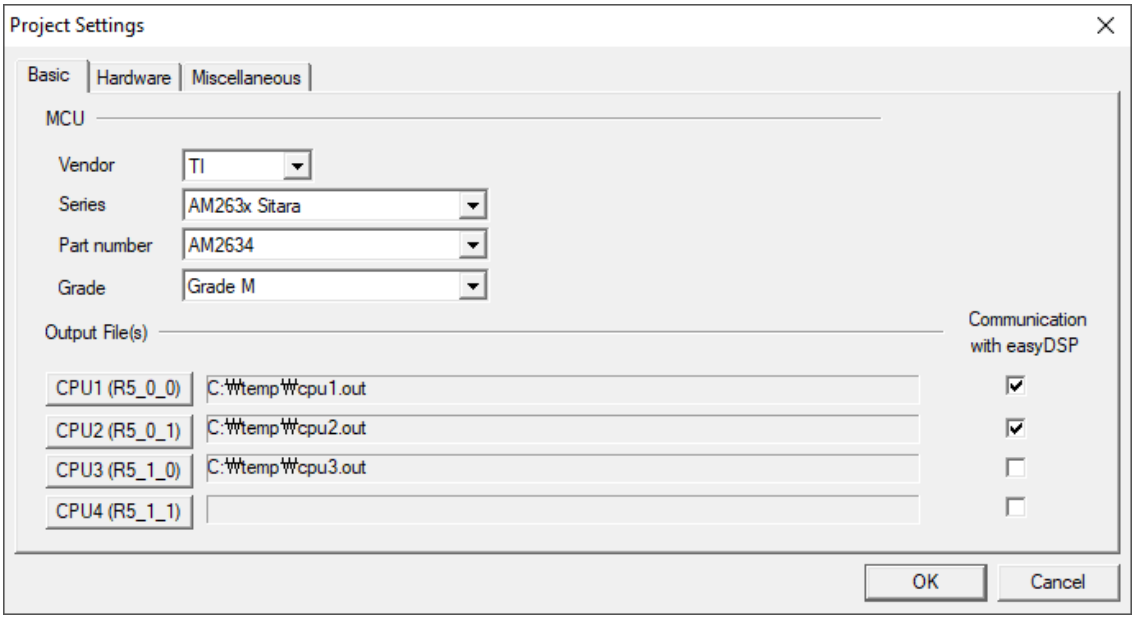
만약 코어 e 와 코어 f 의 easyDSP 프로젝트가 별도의 PC 에서 수행될 때에는 사용자가 코어 f 의 easyDSP 프로젝트에서 메뉴 MCU > Reload *.out 을 직접 수행해야 합니다.

헤더 파일은 하기와 같이 설정합니다.

사용자 프로젝트	노란색 코어
easyAM.h 설정	EASYDSP_POD_IS_CONNECTED_TO_THIS_CORE = **1** EASYDSP_IS_COMMUNICATING_WITH_MULTI_CORES = **0**

Case 3 :

만약 core g, h, c, d 가 각각 CPU1, 2, 3 , 4 일 경우, core g 용 easyDSP 프로젝트 설정은 하기와 같습니다. easyDSP 와 통신하는 코어로 CPU1, CPU2 를 선정하며, 실제 동작하는 모든 코어의 출력 파일을 등록합니다.

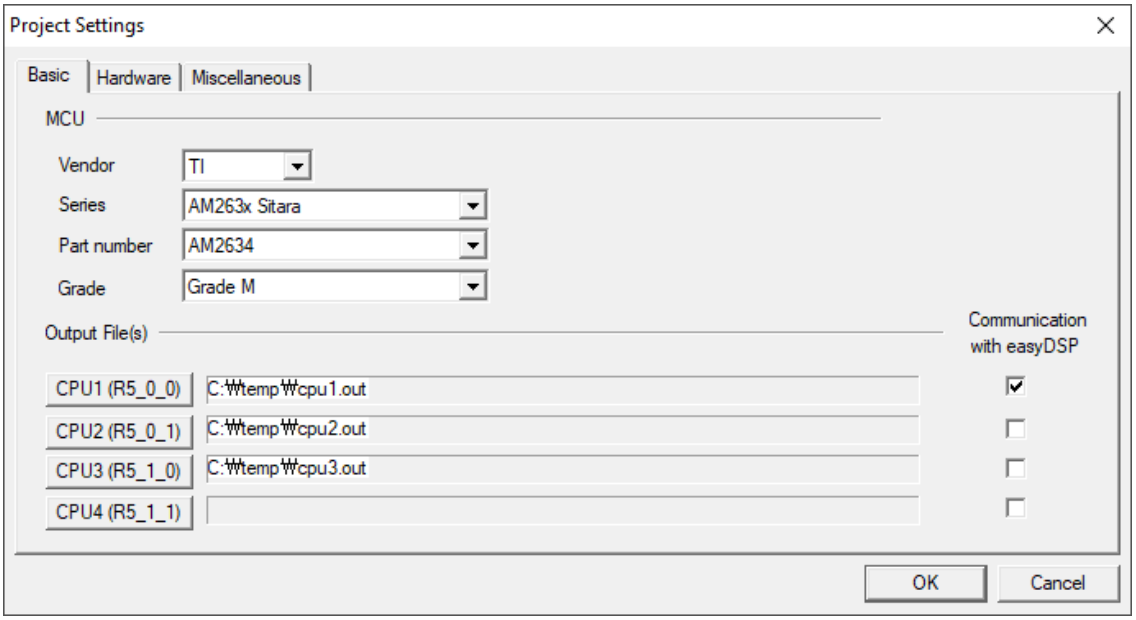


헤더 파일은 하기와 같이 설정합니다.

사용자 프로젝트	노란색 코어
easyAM.h 설정	EASYDSP_POD_IS_CONNECTED_TO_THIS_CORE = **1** EASYDSP_IS_COMMUNICATING_WITH_MULTI_CORES =**1** D_CACHE_IS_ENABLED = **0**

Case 4 :

만약 core i, j, c, d 가 각각 CPU 1, 2, 3 , 4 일 경우, core i 용 easyDSP 프로젝트 설정은 하기와 같습니다. easyDSP 와 통신하는 코어로 CPU1 을 선정하며, 실제 동작하는 모든 코어의 출력 파일을 등록합니다.



헤더 파일은 하기와 같이 설정합니다.

사용자 프로젝트	노란색 코어
easyAM.h 설정	EASYDSP_POD_IS_CONNECTED_TO_THIS_CORE = **1** EASYDSP_IS_COMMUNICATING_WITH_MULTI_CORES =**0**

STEP 4 : 링커 파일 주의 사항

모든 코어에 대해 램 시작번지를 0x7004.0000 또는 이보다 큰 값으로로 설정 바랍니다.
이는 TI 예제와 동일합니다.

```
MEMORY
{
     .
     .
     .

     OCRAM      : ORIGIN = 0x70040000 , LENGTH = 0x40000

     .
     .
     .
}
```

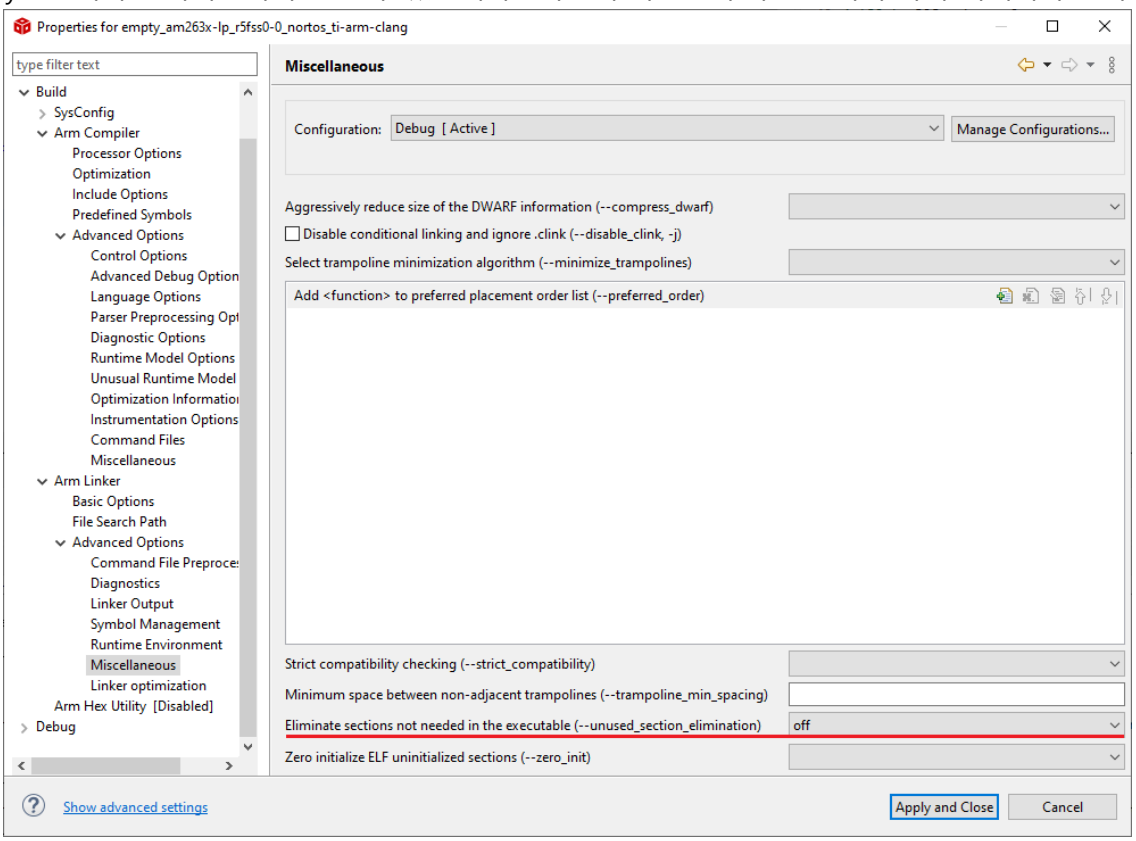
STEP 5 : 변수 이름

주의 사항입니다.
easyDSP 가 한 개의 코어하고만 통신할 경우 사용자가 지정한 변수 이름이 그대로 표시되나, 여러 개의 코어와 통신할 경우 각 코어의 변수 이름을 구분하기 위해 CPUn(n=1,2,3,4)의 경우 'n:변수이름'처럼 사용자가 지정한 변수 이름 앞에 'n:'이 덧붙이게 됩니다.

STEP 6 : IDE 설정

1. **각 코어의 프로젝트를 컴파일할 때마다 각 코어의 rprc 파일이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 개발 환경을 설정 해주세요** (TI CCS 기본 설정). rprc 파일은 램 부팅 및 플래시 프로그래밍할 때 사용됩니다.
2. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.out)에 debug information 이 반드시 포함되어야 합니다 (TI CCS 기본 설정).
3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아

easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수가 포함되게 하기 위해서라면 하기 옵션 설정하세요.



7.4.2 AM263x 하드웨어 설정

easyDSP 와 연결

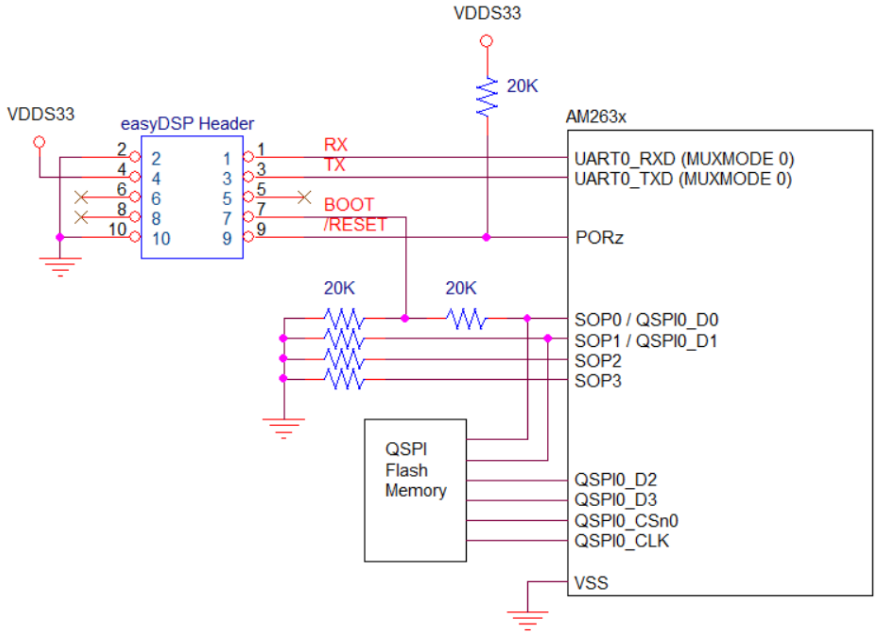
easyDSP 는 MCU 부트 모드를 사용하여, 필요한 부팅 동작을 수행합니다.
램부팅 및 플래시 작업을 위해서는 UART 부트모드를 사용하며, 사용자 프로그램을 수행하기 위해서는 QSPI(4S) - Quad Read Mode 부트모드를 사용합니다.

Boot Mode	SOP3	SOP2	SOP1	SOP0
QSPI (4S) - Quad Read Mode	0	0	0	0
UART	0	0	0	1
QSPI (1S) - Single Read Mode	0	0	1	0
QSPI (4S) - Quad Read UART Fallback Mode	0	1	0	0
QSPI (1S) - Single Read UART Fallback Mode	0	1	0	1
DevBoot	1	0	1	1

easyDSP 가 MCU 리셋시 부트핀을 적절히 설정하기 위해 BOOT 핀을 SOP0 핀에 연결하며, SOP1, SOP2, SOP3 는 Low 로 고정합니다.

easyDSP 헤더 RX, TX 핀은 MCU UART0 (MUXMODE 0)에 연결해주세요. 다른 UART 채널 및 핀에 연결될 경우 램부팅과 플래시 프로그래밍이 지원되지 않으며 따라서 BOOT 핀, /RESET 핀도 연결하지 마세요.
또한 QSPI0 에 플래시를 연결합니다. 플래시는 Sector Erase 명령이 64kB block 에 적용되는 제품으로 사용하셔야 합니다. TI 평가보드에서 사용되는 S25FL128SAGNFI000 제품을 권장 드립니다.

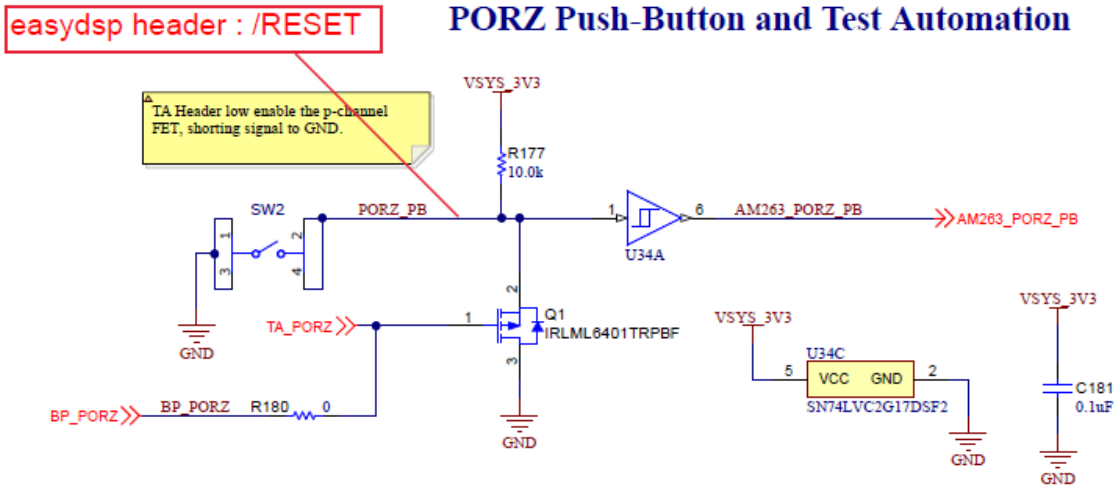
easyDSP 헤더 4 번 핀에는 MCU 의 VDDS33 를 연결하여 주십시요.

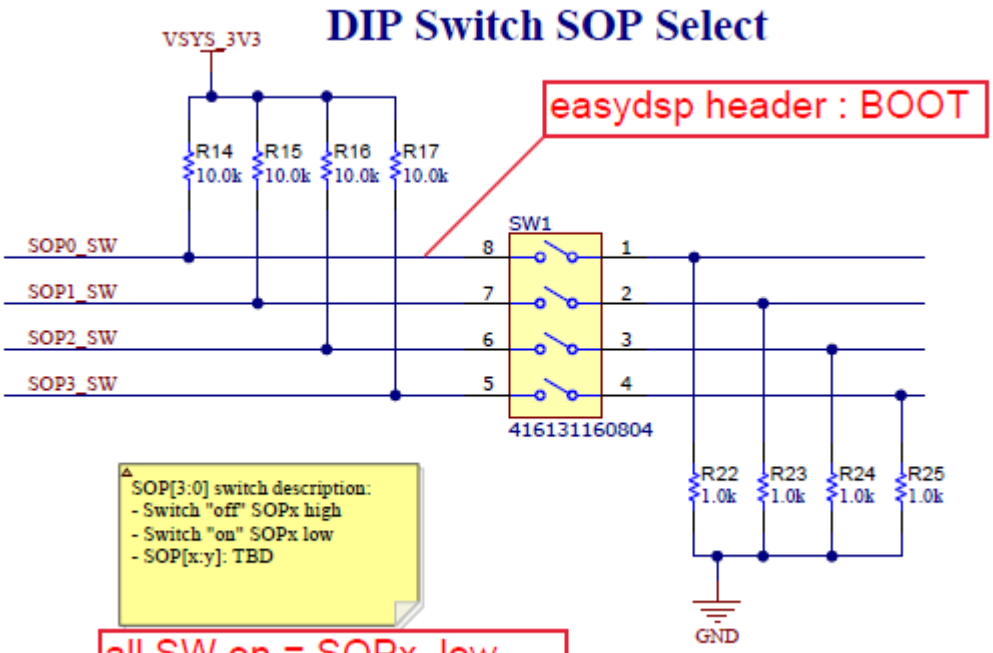


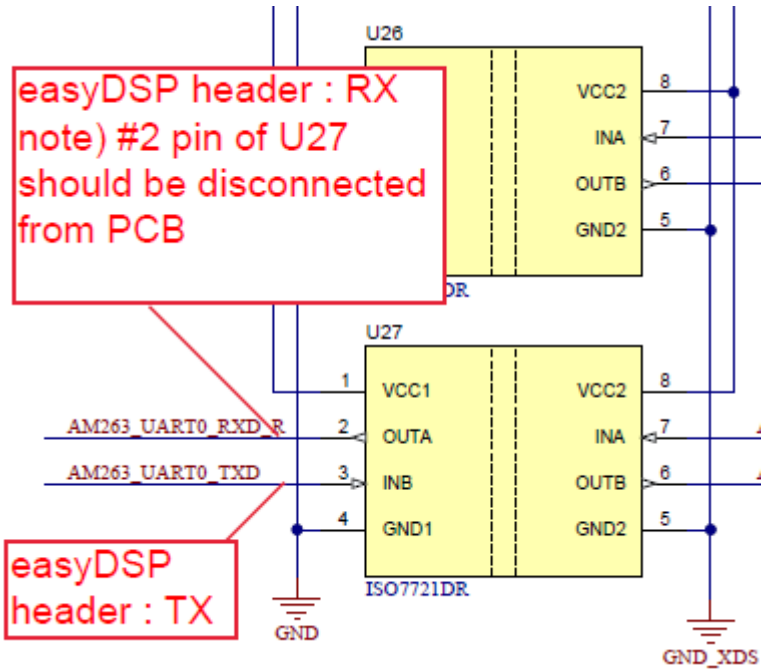
기타 주의 사항 :
- 25MHz XTAL 클럭소스를 사용
- MCU 리셋 (PORz)이 해제된 후 약 1msec 뒤에 MCU 는 SOP 핀의 신호를 취득하여 부트 모드를 결정합니다. 따라서 SOP 핀에 연결되는 다른 회로가 리셋 이후 일정 시간 동안 (약 2msec) 신호를 출력하지 않도록 보드 설계가 되어야 합니다.
- easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업되어 있습니다.
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU PORz 신호사이에 리셋 IC 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초 내에 PORz 에 전달해야 함

AM263x Launchpad 에 easyDSP 연결

TI AM263x Launchpad 에 easyDSP 를 연결하기 위한 수작업에 대해 하기 참조하세요.
SW1 은 모두 ON 이 되어야 하며, U27 의 2 번 핀은 PCB 에서 분리되어야 함에 유의하세요.


easydsp header : /RESET
PORZ Push-Button and Test Automation
TA Header low enable the p-channel FET, shorting signal to GND.
VSYS_3V3
R177
10.0k
SW2
PORZ_PB
U34A
AM263_PORZ_PB
AM263_PORZ_PB
GND
TA_PORZ
Q1
IRLML6401TRPBF
VSYS_3V3
U34C
VCC
GND
SN74LVC2G17DSF2
VSYS_3V3
C181
0.1uF
BP_PORZ
BP_PORZ
R180
0
GND
GND
GND


DIP Switch SOP Select
VSYS_3V3
easydsp header : BOOT
R14
10.0k
R15
10.0k
R16
10.0k
R17
10.0k
SW1
SOP0_SW
SOP1_SW
SOP2_SW
SOP3_SW
416131160804
R22
1.0k
R23
1.0k
R24
1.0k
R25
1.0k
GND
SOP[3:0] switch description:
- Switch "off" SOPx high
- Switch "on" SOPx low
- SOP[x:y]: TBD
all SW on = SOPx low


easyDSP header : RX
note) #2 pin of U27
should be disconnected
from PCB
U26
VCC2
INA
OUTB
GND2
U27
VCC1
OUTA
INB
GND1
VCC2
INA
OUTB
GND2
AM263_UART0_RXD_R
AM263_UART0_TXD
ISO7721DR
easyDSP
header : TX
GND
GND_XDS

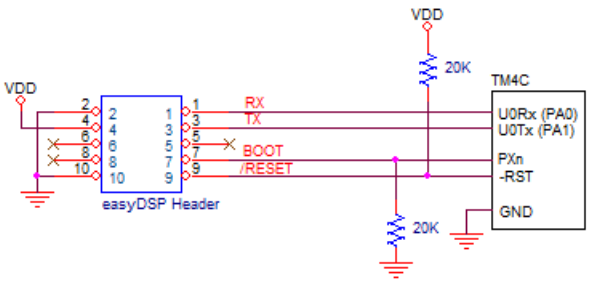
7.5 TM4C

TM4C 설정

STEP 1 : 하드웨어 설정

easyDSP 는 플래시 라이팅을 위해 MCU 내장 롬 부트로더를 사용합니다. 따라서 롬 부트로더에서 지원하는 UART0 채널을 사용해야 합니다.
다른 채널을 사용할 경우 easyDSP 는 모니터링은 지원하나, 플래시 라이팅은 지원하지 않으며, 또한 easyDSP 소스파일 easyTM4C.c 를 해당 채널에 맞게 직접 수정하셔야 합니다.
하기 그림과 같이 연결하시기 바랍니다.
PXn 포트는 easyDSP 에서 부트핀으로 사용되며 사용자가 easyTM4C.h 에서 설정 가능합니다. 단, 하기 조건 참조하세요.

1. TM4C129x 에서는 PC0-3, PD7, PE7 은 부트핀으로 사용할 수 없음.
2. TM4C123x 에서는 PC0-3, PD7, PF0 은 부트핀으로 사용할 수 없음.
3. 해당 부트핀에 연결된 다른 회로가 있을 경우, MCU 리셋이후 약 1 초동안은 이 회로에서 출력 신호를 발생시키면 안됨.



기타 주의 사항 :
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU -RST 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 -RST 에 전달해야 함
- easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업되어 있습니다.

STEP 2 : easyDSP 제공 헤더 파일 수정

먼저 easyDSP 통신을 위해 제공되는 소스파일 (easyTM4C.h, easyTM4C.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩TM4C 에서 찾을 수 있습니다.

사용 환경에 맞춰 ezsyTM4C.h 초기 부분을 설정하시기 바랍니다.
타켓 MCU, MCU 클럭 주파수, easyDSP 통신 보드레이트 및 부트핀을 설정하시기 바랍니다. easyDSP

프로젝트에서 사용될 보드레이트와 동일하게 설정합니다.

```
//*****************************************************************************
// step 1 : set target MCU
//          if TM4C129x is used, set EZ_USE_TM4C129x as 1
//          if TM4C123x is used, set EZ_USE_TM4C123x as 1
//*****************************************************************************
#define EZ_USE_TM4C129x     0
#define EZ_USE_TM4C123x     1

//*****************************************************************************
// step 2 : set the system clock frequency
//          for example, 120000000L for TM4C129x, 80000000L for TM4C123x
//*****************************************************************************
#define EZ_SYS_CLK_FREQ     80000000L

//*****************************************************************************
// step 3 : set the baud rate for UART communication with easyDSP
//          it should be same to the baudrate of easyDSP project
//*****************************************************************************
#define EZ_BUAD_RATE        230400

//*****************************************************************************
// step 4 : boot pin (PXn) selection
//          don't use PC0-3, PD7, PE7 for TM4C129x
//          don't use PC0-3, PD7, PF0 for TM4C123x
//          below example sets PB5 as a boot pin
//*****************************************************************************
#define EZ_SYSCTL_PERIPH_GPIOX  SYSCTL_PERIPH_GPIOB
#define EZ_GPIO_PORTX_BASE      GPIO_PORTB_BASE
#define EZ_GPIO_PIN_n           GPIO_PIN_5
```

STEP 3 : easyDSP 관련 함수 호출

먼저 main.c 상단에 easyTM4C.h 를 include 하여 주시고, main 함수 가장 첫 부분에 easyDSP_boot() 함수를 호출하고, 기타 초기화 루틴 이후 easyDSP_init() 함수를 호출하시기 바랍니다.
easyDSP_boot() 함수에서는 부트핀의 상태를 기반으로 플래시 프로그래밍을 위해 MCU 내장 롬부트로더에 진입할지 여부를 판단합니다. easyDSP 의 플래시 프로그래밍 기능을 사용하지 않는다면 본 함수도 필요 없습니다.
easyDSP_init() 함수에서는 easyDSP 와의 통신을 위한 각종 설정을 수행합니다.

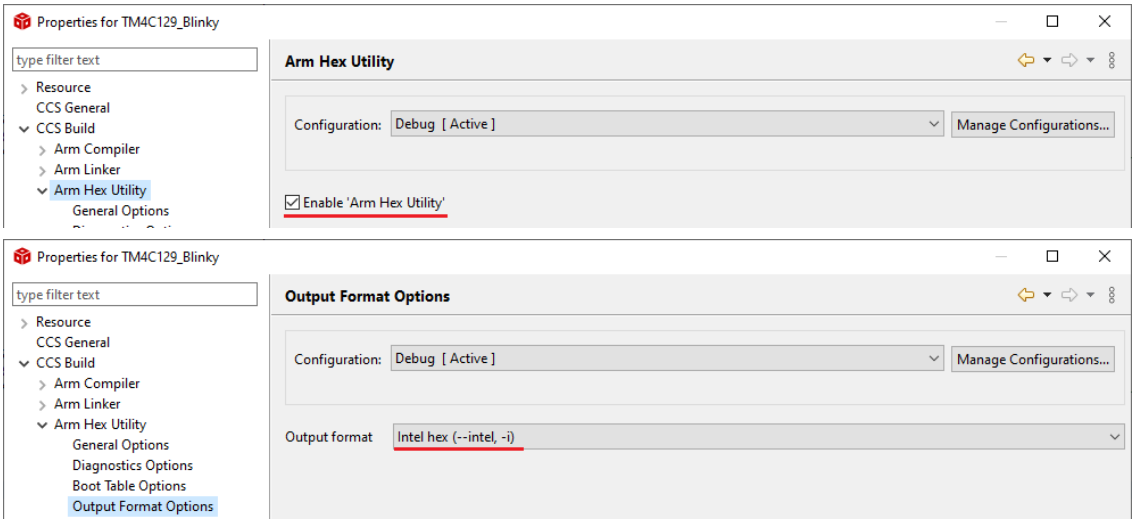
```c
#include "easyTM4C.h"

int main(void)
{
    // the very beginning, call easyDSP_boot() to enable ROM boot loader if required
    easyDSP_boot();

    // initial setting
    .
    .
    .
    .
    .
    .

    // call easyDSP_init() to enable easyDSP monitoring
    easyDSP_init();

    // loop forever
    while(1)
    {
        .
        .
        .
    }
}
```

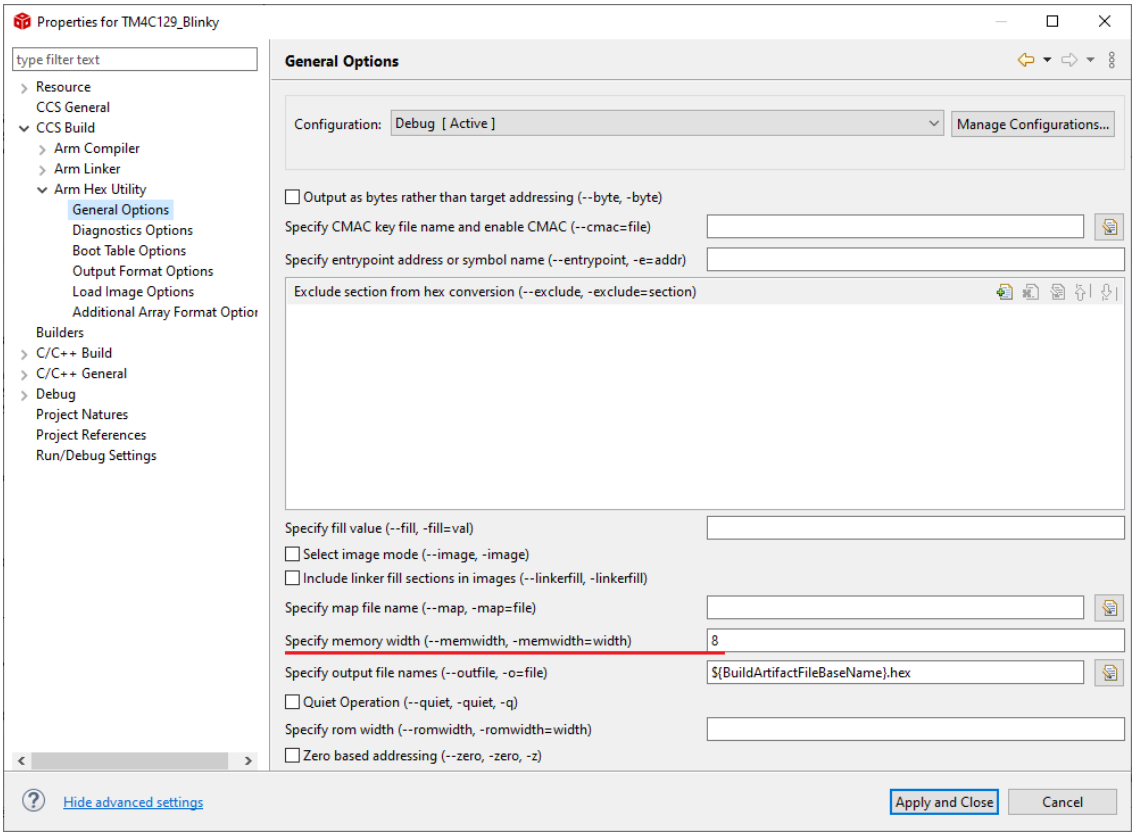
STEP 4 : IDE 설정

1. 매 컴파일마다 hex 파일 (인텔 형식) 이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 IDE 를 설정해주세요. Hex 파일은 플래시 프로그래밍할 때 사용됩니다.

Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.

하기 CCS 경우 참조하세요. 특히 CCS 사용시 memory width 옵션을 8 로 설정함에 유의하세요.





2. easyDSP 로 모니터링을 수행하기 위해서는, 출력 파일 (예:*.out)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수를 포함되게 하기 위해서라면 해당하는 IDE 세팅이 필요합니다.

4. easyTM4C.c 에서 인라인 함수를 사용하므로, 필요시 컴파일러 옵션 c99 모드를 활성화시켜 주세요.

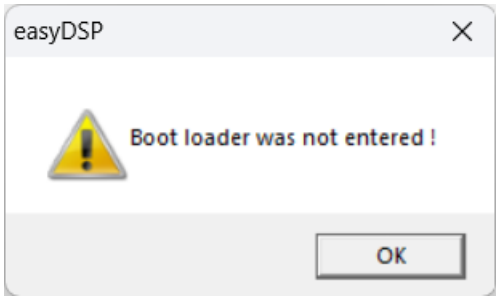
STEP 5 : 기타 설정

1. easyDSP 가 플래시 프로그래밍을 하기 위해서는 MCU 의 플래시 보호 설정이 없어야 합니다.

2. BOOTCFG 레지스터의 EN 비트는 1 이어야 합니다. 이로서 MCU 는 easyDSP_boot()에 의해 부팅 방식을 결정하게 됩니다.

3. easyDSP 가 플래시 프로그래밍하기 위해서는, 플래시가 전부 지워져 있거나, easyDSP 소스파일이 이미 플래시에 프로그래밍이 되어 있어야 합니다.

다른 상황 (예 : easyDSP 소스파일이 적용되지 않은 프로그램이 플래시에 존재)에서는 플래시 프로그래밍이 지원되지 않으며 하기와 같은 에러 메시지가 송출되오니, 디버거를 사용한 플래시 프로그래밍이 필요합니다.

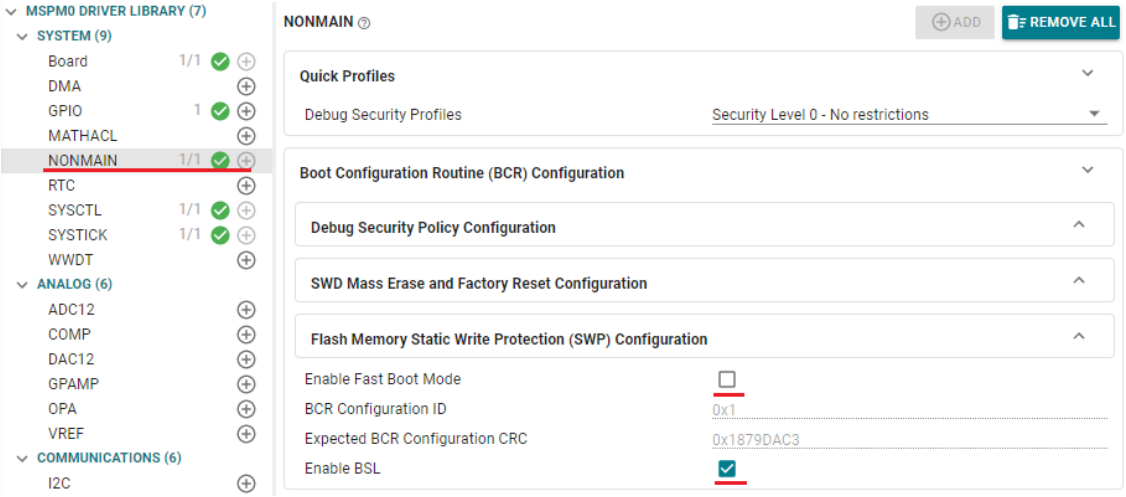


7.6 MSPM0

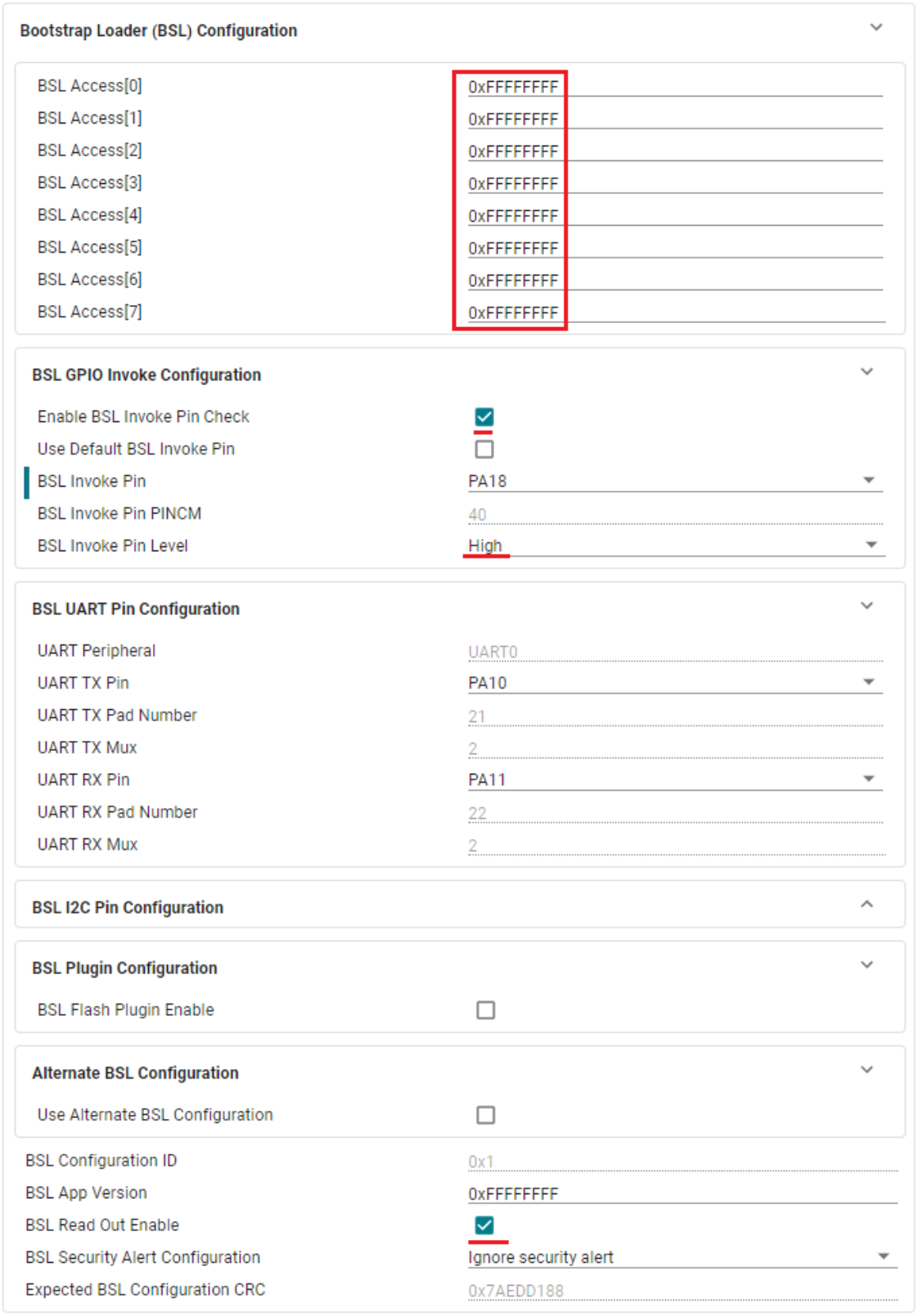
STEP 1 : SysConfig 설정 - NONMAIN 부분

easyDSP 는 TI 가 제공하는 SysConfig 기반 생성된 코드를 사용합니다. 하기에서 SysConfig 를 설정하는 방법을 SysConfig 1.16.1 기준으로 설명드립니다.
먼저 NONMAIN 부분입니다. BCR, BSL 관련 설정을 할 수 있습니다.
만약 공장 출하 초기값을 사용하시려면 STEP 1 은 생략하셔도 됩니다. 그렇지 않다면 하기를 참조하세요.

먼저 BCR Configuration 부분입니다.
Enable Fast Boot Mode 는 허용하지 않으며, Enable BSL 은 허용되어야 합니다.



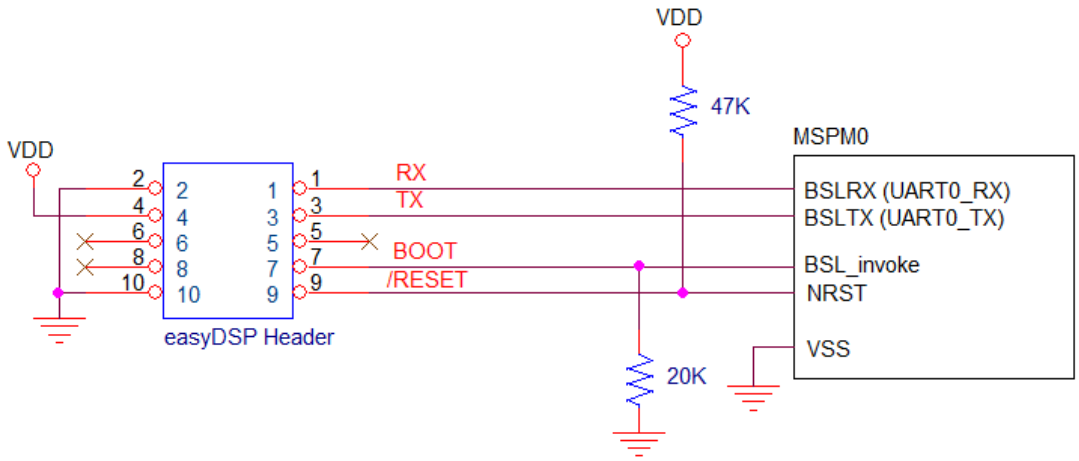
다음으로 BSL Configuration 부분입니다.
필요시 BSL 에 진입하기 위한 비밀번호 32 바이트를 설정합니다. 초기값은 32 바이트 모두 0xFF 입니다.
Enable BSL Invoke Pin Check 는 활성화하고,
Default BSL Invoke Pin 을 사용할 수 있으며 필요시 BSL Invoke Pin 을 설정하실 수 있습니다. 단, BSL Invoke Pin Level 은 반드시 High 로 설정합니다.
필요시 BSL UART Pin 을 설정하실 수 있습니다.
마지막으로 BSL Read Out Enable 을 활성화합니다.



주의 사항이 있습니다. NONMAIN 영역의 플래시는 easyDSP 에서 프로그래밍할 수 없습니다. 따라서 변경된 부분은 디버거나 다른 툴로 플래시 프로그래밍이 되어야 합니다.

STEP 2 : 하드웨어 구성

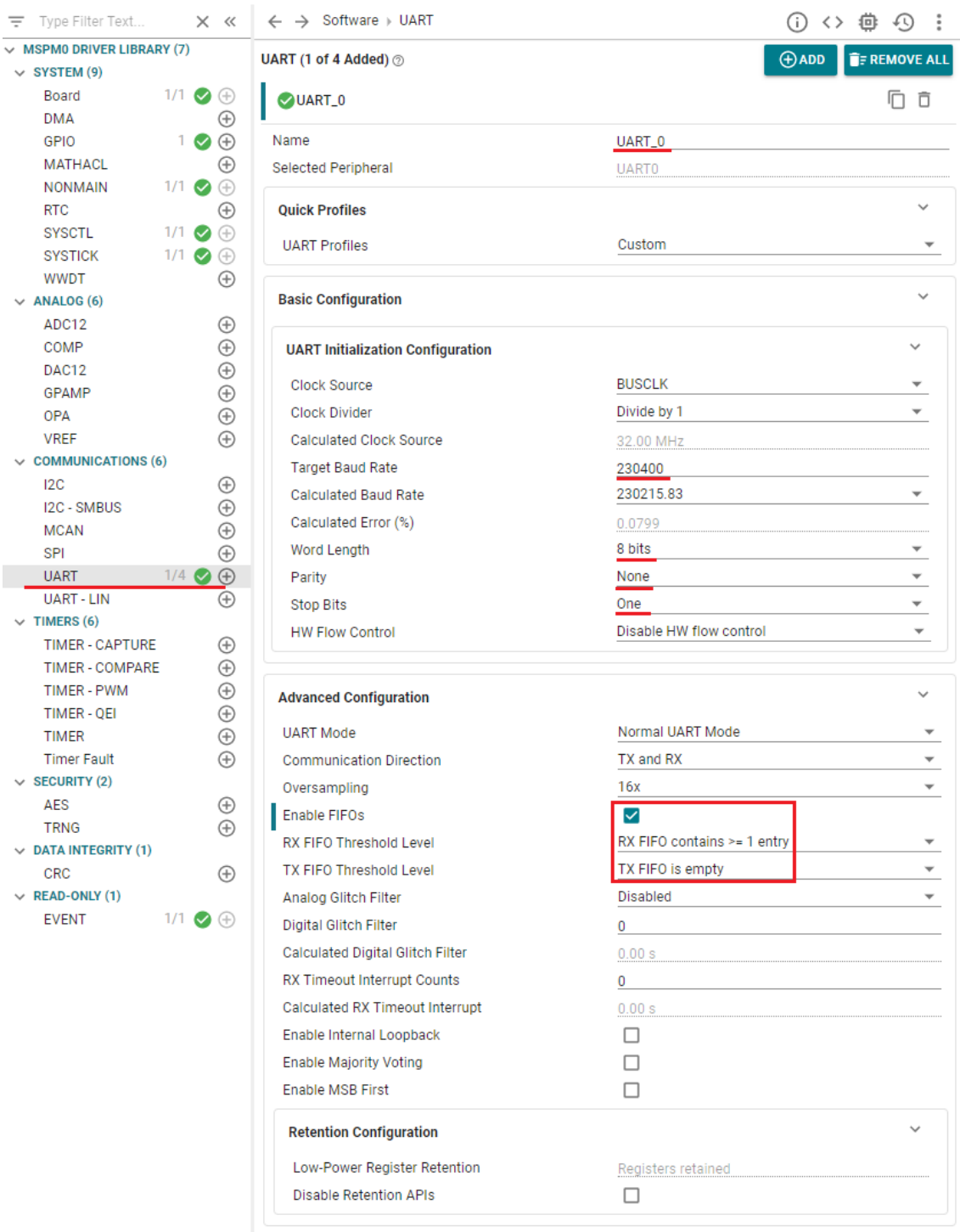
easyDSP 와 MCU 간의 결선은 하기와 같습니다.
상기 STEP1 에서 설정된 값으로 BSL_invoke, BSLRX, BSLTX 핀을 연결하거나,
STEP 1 을 생략하고 공장 출하 기준 초기값을 사용할 경우, BSL_invoke, BSLRX, BSLTX 핀은 MCU 종류 및 BSL configuration 에 따라 다르므로 타켓 MCU 의 데이타시트를 참조하시기 바랍니다.
예를 들어, 공장 출하 초기 기준으로, MSPM0L1306xRHB 의 경우 BSLRX 는 26 번핀 , BSLTX 는 27 번핀, BSL_invoke 은 22 번핀이며, MSPM0G3507SPM 의 경우 BSLRX 는 57 번핀 , BSLTX 는 56 번핀, BSL_invoke 은 11 번핀입니다.
참고로 BSL_RX, BSL_TX 는 UART0 으로 사용됩니다.



기타 주의 사항 :
- /RESET 핀은 MCU NRST 에 직결
- easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업되어 있습니다.

STEP 3 : SysConfig 설정 - UART 부분

BSL_RX, BSL_TX 핀은 항상 UART0 이므로, UART_0 이름으로 UART0 채널을 생성합니다.
보드레이트를 선정합니다. 선정된 값은 easyDSP 프로젝트에서 선정된 보드레이트와 동일해야 합니다.
UART 통신 규격은 8 비트, no parity, one stop bit 이며
FIFO 는 활성화하며 FIFO Threshold Level 은 아래와 같이 설정합니다.



인터럽트는 Receive, Transmit 를 활성화시키며 최하위 우선순위로 설정합니다. TX/RX 핀의 풀업 저항을 활성화시키며,

UART0 의 RX/TX 핀을 설정합니다. 하기 그림에서는 MSPM0G3507SPM 의 출하 기준으로 BSLRX 는 57 번핀 , BSLTX 는 56 번핀으로 할당하였습니다.

Extend Configuration

Enable Extend Features ☐

Interrupt Configuration

Enable Interrupts: Receive, Transmit

Interrupt Priority: Level 3 - Lowest

DMA Configuration

Configure DMA RX Trigger: None

Configure DMA TX Trigger: None

Pin Configuration

TX Pin

Direction: Output

IO Structure: High-Drive

Enable pin configuration ☑

Digital IOMUX Features

Internal Resistor: Pull-Up Resistor

Invert: Disabled

Drive Strength Control: High

High-Impedance: Disabled

RX Pin

Direction: Input

IO Structure: High-Drive

Enable pin configuration ☑

Digital IOMUX Features

Internal Resistor: Pull-Up Resistor

Invert: Disabled

Hysteresis Control: Disabled

Wakeup Logic: Disabled

PinMux Peripheral and Pin Configuration

UART Peripheral: UART0

RX Pin: PA11/57

TX Pin: PA10/56

Other Dependencies

STEP 4 : easyDSP 제공 소스 파일

easyDSP 는 TI 가 제공하는 driverlib 라이브러리를 사용합니다. 사용자 코드 프로젝트를 생성할 때, driverlib 를 포함하여 주세요.

먼저 easyDSP 통신을 위해 제공되는 소스파일 (easyMSPM0.h, easyMSPM0.c)을 프로젝트에 포함하시기 바랍니다. 해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩MSPM0 에서 찾을 수 있습니다.

먼저 main.c 상단에 easyMSPM0.h 를 include 하여 주시고,
main 함수 적절 부분에 easyDSP_init() 함수를 호출하시기 바랍니다.

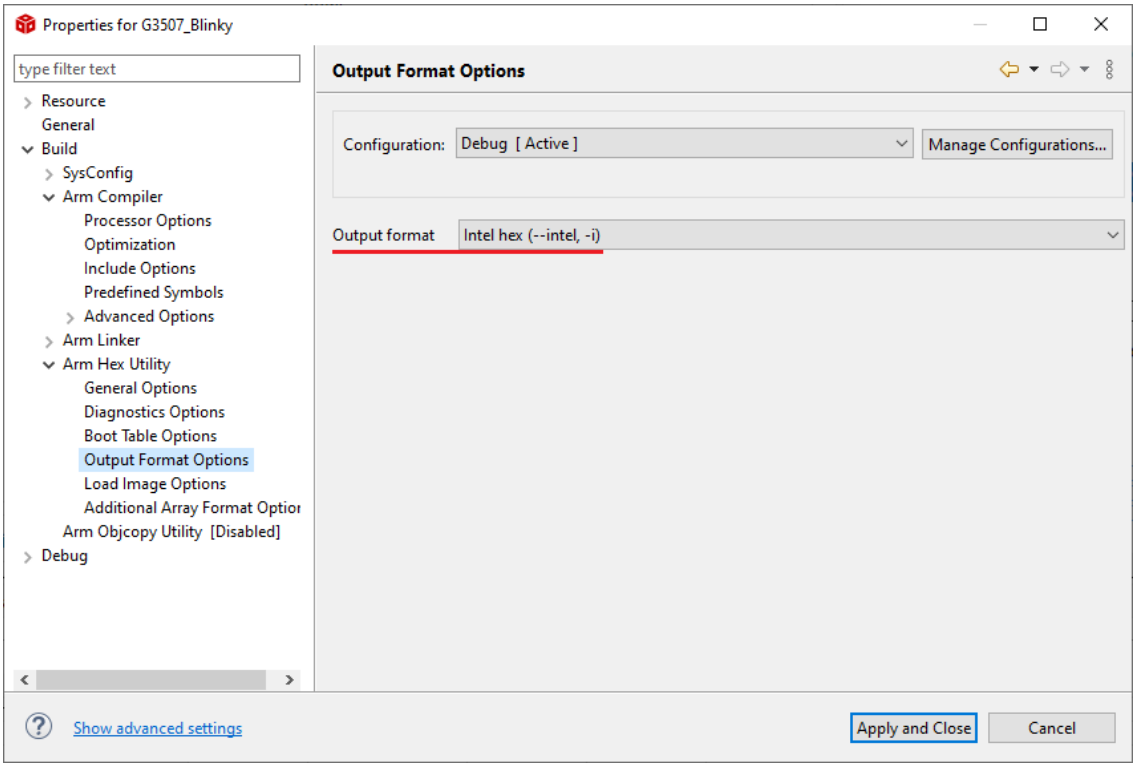
```
#include "easyMSPM0.h"

int main(void)
{
    // initial setting
    .
    .
    .
    .
    .
    .

    // call easyDSP_init() to enable easyDSP monitoring
    easyDSP_init();

    // loop forever
    while(1)
    {
        .
        .
        .
    }
}
```

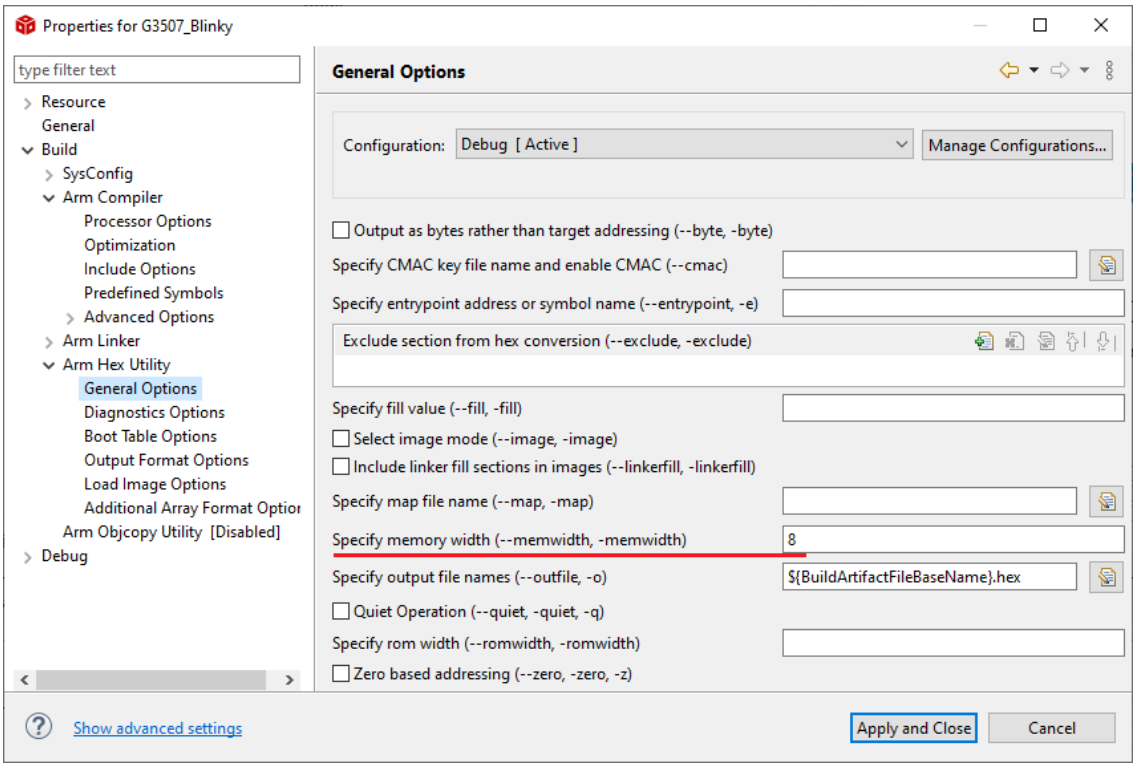
STEP 5 : IDE 설정

1. 매 컴파일마다 hex 파일 (인텔 형식) 이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 IDE 를 설정해주세요. Hex 파일은 플래시 프로그래밍할 때 사용됩니다.

Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.

하기 CCS 경우 참조하세요. 특히 CCS 사용시 memory width 옵션을 8 로 설정함에 유의하세요.



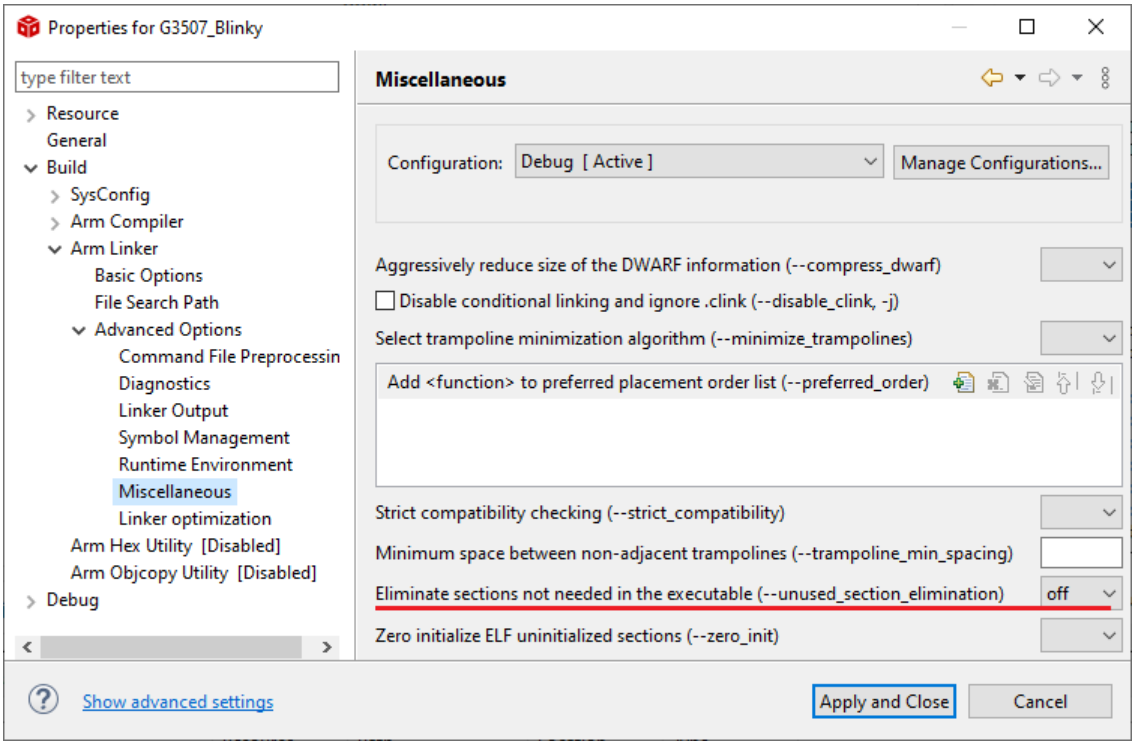


2. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

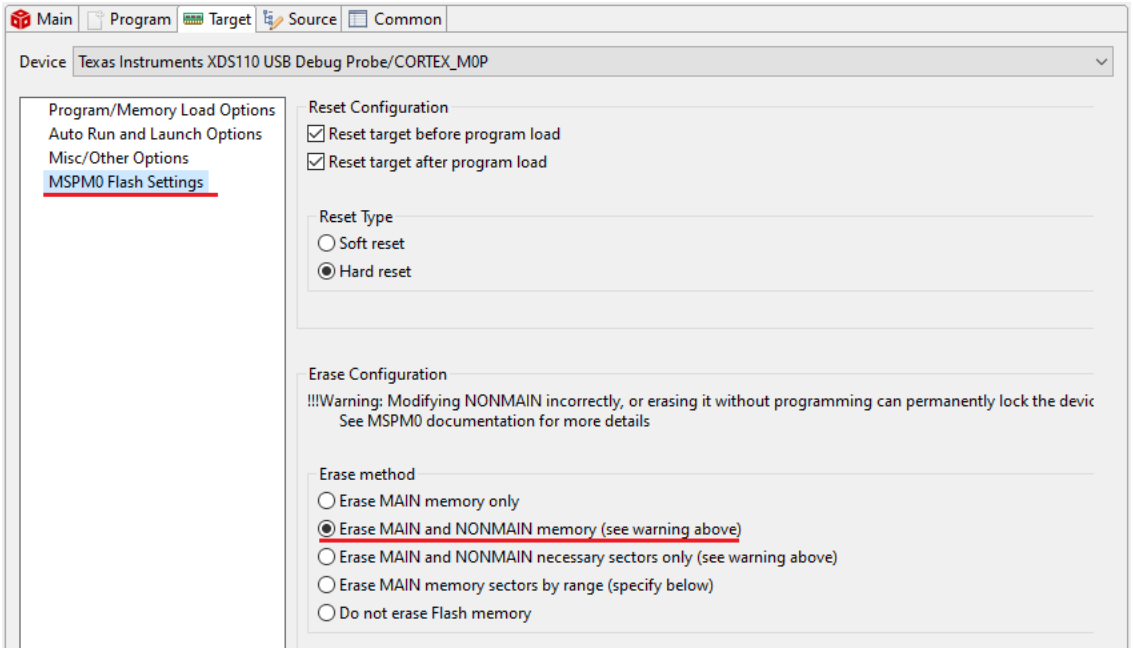
3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다.

이 경우에도 변수를 포함되게 하기 위해서라면, 최적화 수준을 낮추거나, 예를 들어 CCS 라면 링커 옵션을 하기와 같이 설정하세요.

다른 개발 환경이라면 적절히 해당하는 세팅이 필요합니다.



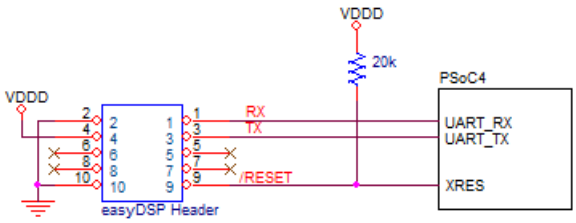
4. STEP 1 에서 변경된 NONMAIN 영역 플래시를 디버거로 프로그래밍할 경우, CCS 에서 하기 옵션 설정이 필요합니다.



7.7 PSoC4

7.7.1 PSoC4 하드웨어 설정

상기 "PSoC4 소프트웨어 설정"에서 선정된 UART 핀을 easyDSP 와 연결합니다 .
easyDSP 헤더 4 번핀에는 MCU 의 VDDD 를 연결하여 주십시요 .
RX, TX 신호는 easyDSP 포드 내에서 100k 오옴으로 풀업되어 있습니다.



기타 고려 사항 :

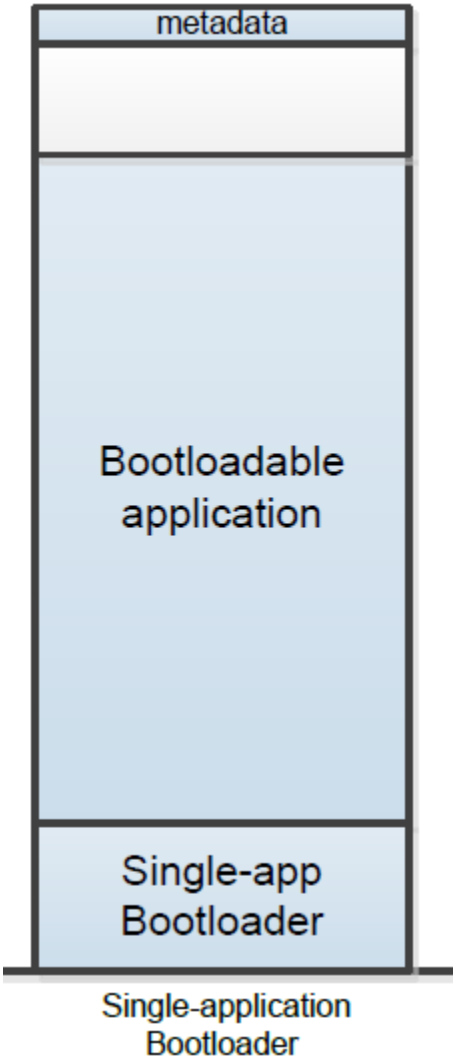
- TX, RX 선은 MCU 의 해당 핀으로 직접 연결
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU XRES 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 XRES 에 전달해야 함
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함

7.7.2 PSoC4 소프트웨어 설정

---

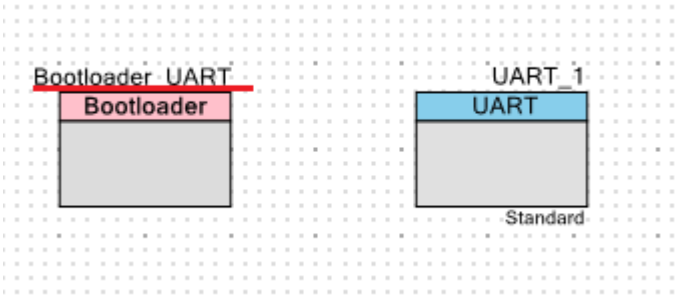
easyDSP 는 플래시 프로그래밍을 지원하기 위해 Single-application bootloader 구성만을 지원합니다. 다른 구성에서는 변수 모니터링은 가능하나 플래시 프로그래밍이 불가능함에 유의 바랍니다.
하기에 easyDSP 를 사용하기 위한 소프트웨어 설정을 PSoC Creator 4.4 기준으로 설명합니다.

부트로더 및 부트로더블 프로젝트를 작성하는 방식 자체는 업체의 매뉴얼을 참조하세요.



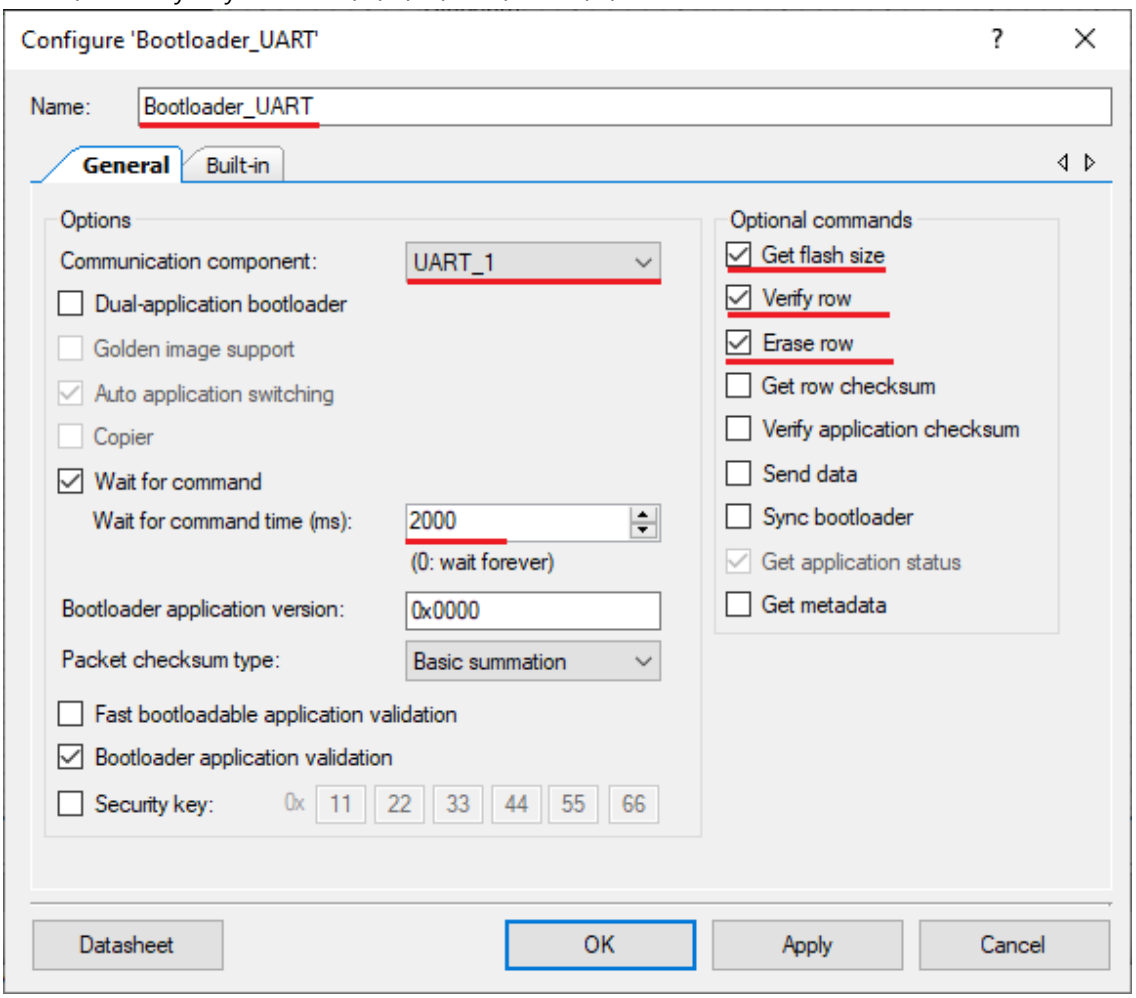
STEP 1 : Bootloader 프로젝트 작성

먼저 컴포넌트 카탈로그에서 하기와 같이 schematic 을 작성합니다.
Bootloader 컴포넌트의 이름을 Bootloader_UART 로 변경합니다.
MCU 부하를 크게 주지 않는 이상 다른 컴포넌트 추가도 가능합니다 (예 LED)



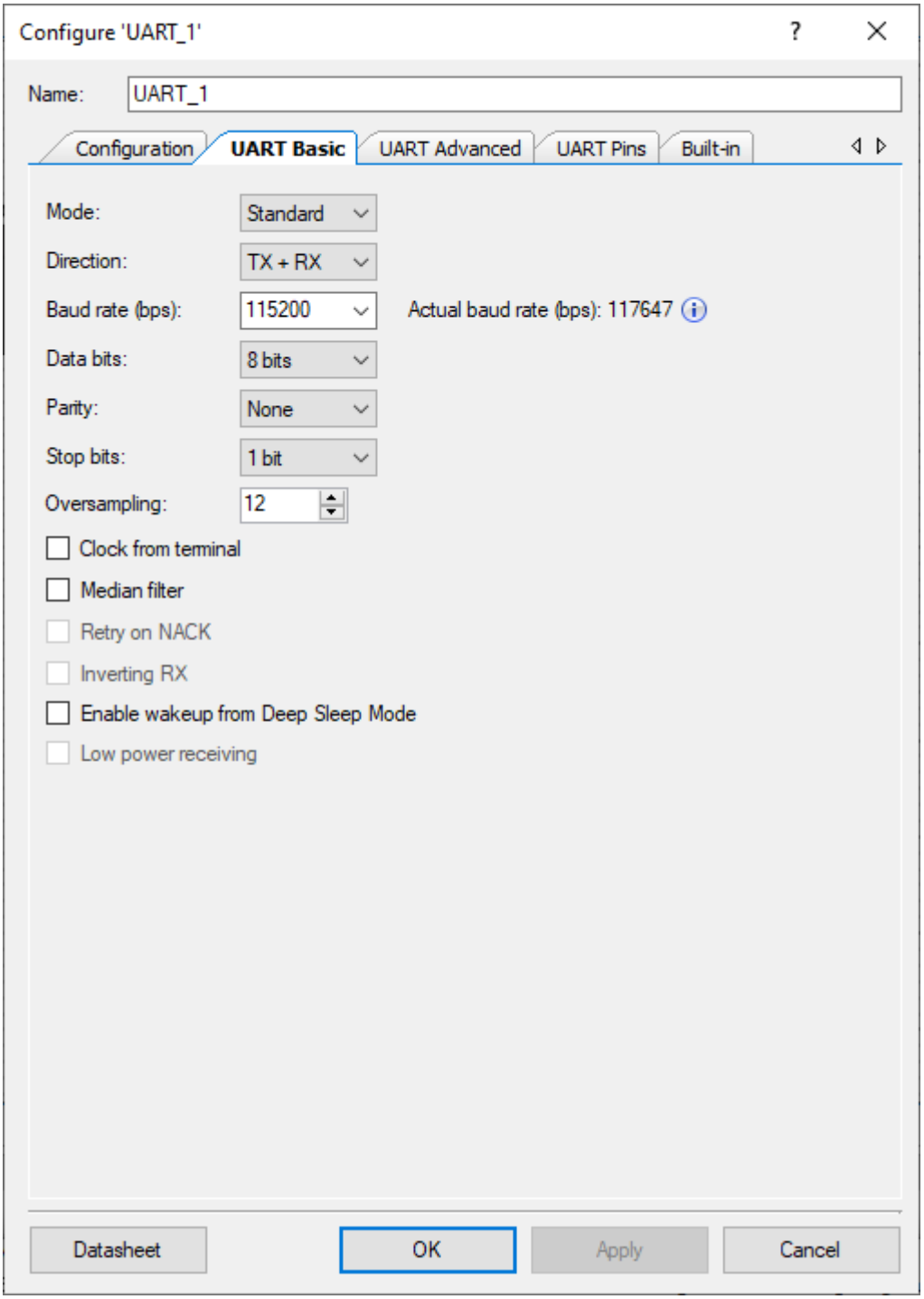
먼저 Bootloader_UART 컴포넌트 설정은 다음과 같습니다.
Communication component 을 UART_1 으로 설정하고, Optional commands 를 설정합니다.
나머지는 디폴트값을 사용하세요.

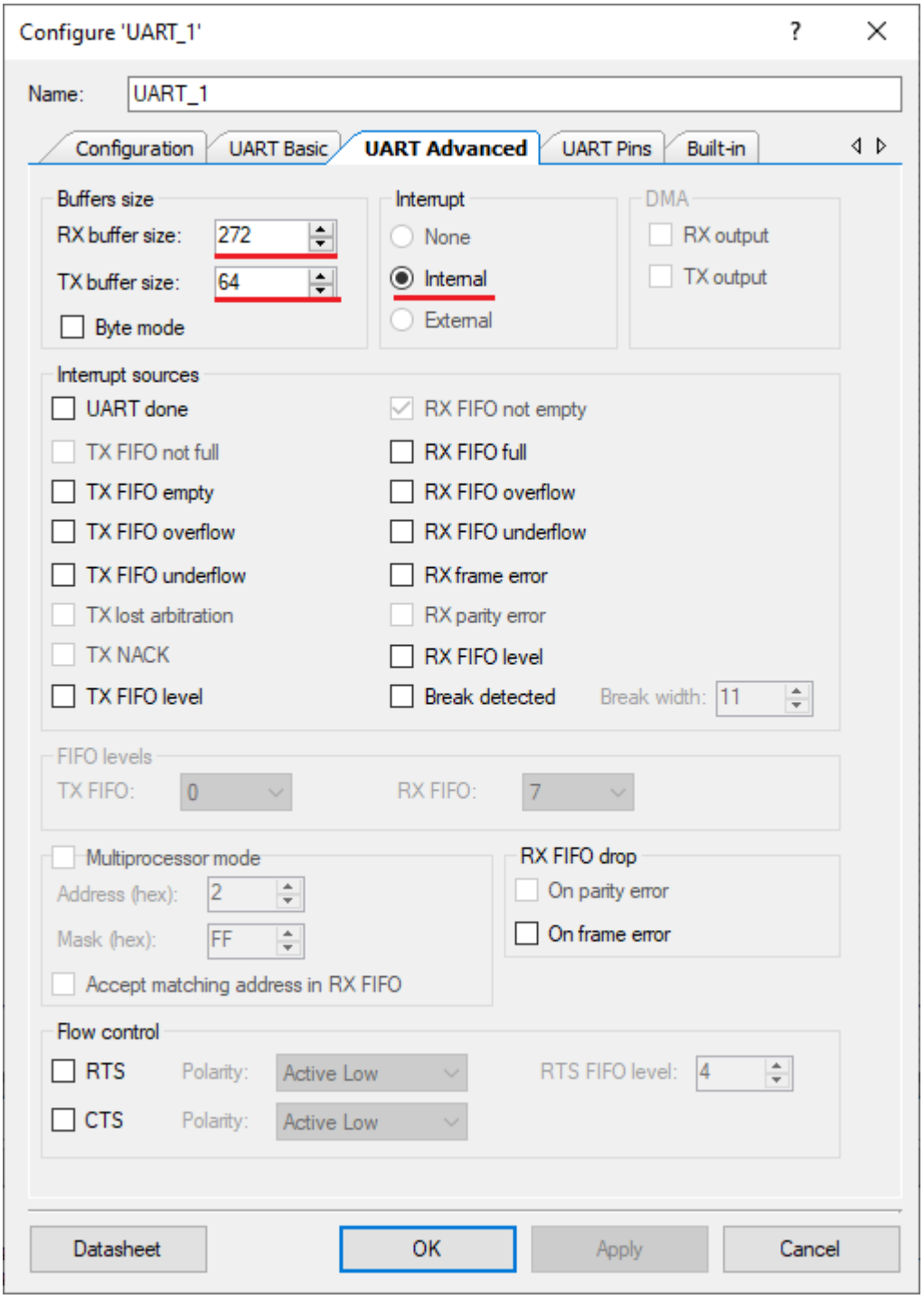
필요시 Security key 를 활성화하여 사용 가능합니다.



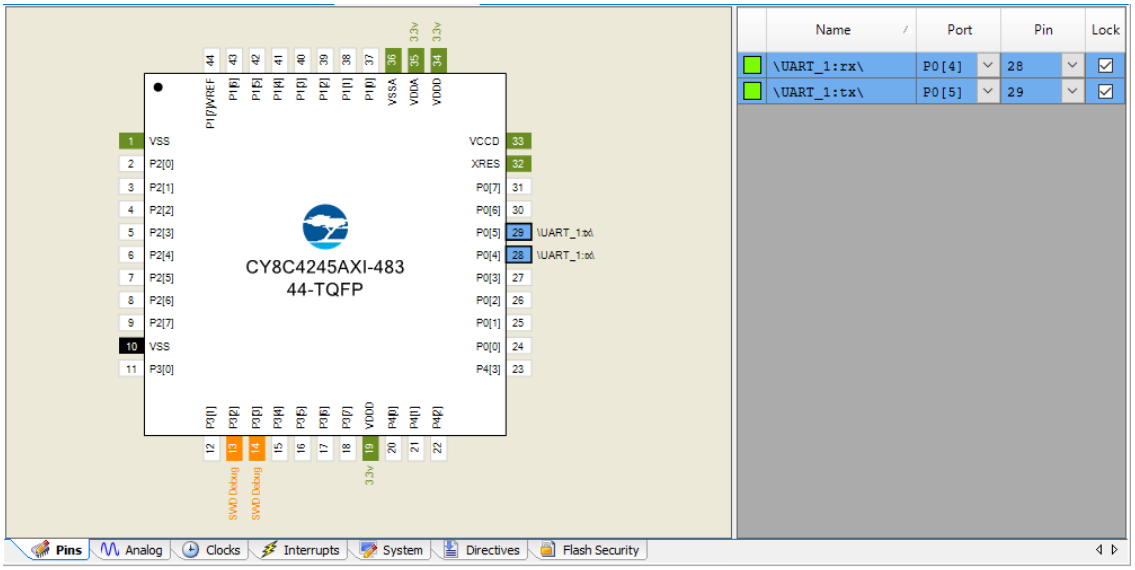
UART 컴포넌트에서 UART Basic 탭은 디폴트 그대로 사용합니다. 통신 방식은 115200bps, 8bits, one stop, no parity 로 고정입니다.

UART Advanced 탭은 Buffer size 를 변경함에 주의하세요.


Configure 'UART_1'
Name:
UART_1
Configuration
UART Basic
UART Advanced
UART Pins
Built-in
Mode:
Standard
Direction:
TX + RX
Baud rate (bps):
115200
Actual baud rate (bps): 117647
Data bits:
8 bits
Parity:
None
Stop bits:
1 bit
Oversampling:
12
Clock from terminal
Median filter
Retry on NACK
Inverting RX
Enable wakeup from Deep Sleep Mode
Low power receiving
Datasheet
OK
Apply
Cancel



UART 핀을 사용자 보드에 맞게 선정하시기 바랍니다. 본 예제에서는 P0.4, P0.5 에 할당되었습니다.



마지막으로 main() 함수 상단에 Bootloader_UART_Start() 호출하시기 바랍니다.
이로서 easyDSP 사용을 위한 부트로더 프로젝트가 완료되었습니다.

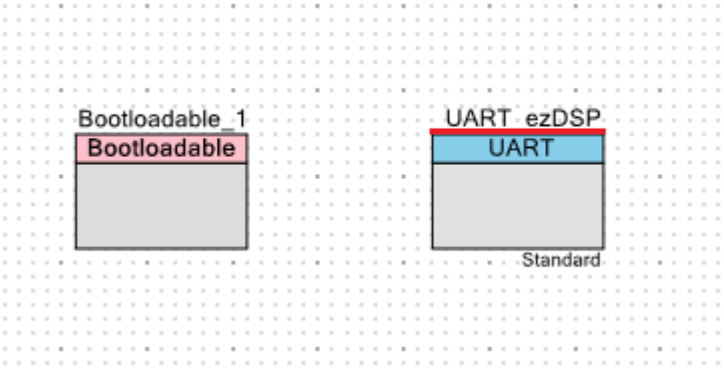
```
int main(void)
{
    Bootloader_UART_Start();
    //CyGlobalIntEnable; /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g. MyInst_Start()) */

    for(;;)
    {
        /* Place your application code here. */
    }
}
```

STEP 2 : Bootloader 프로젝트 굽기

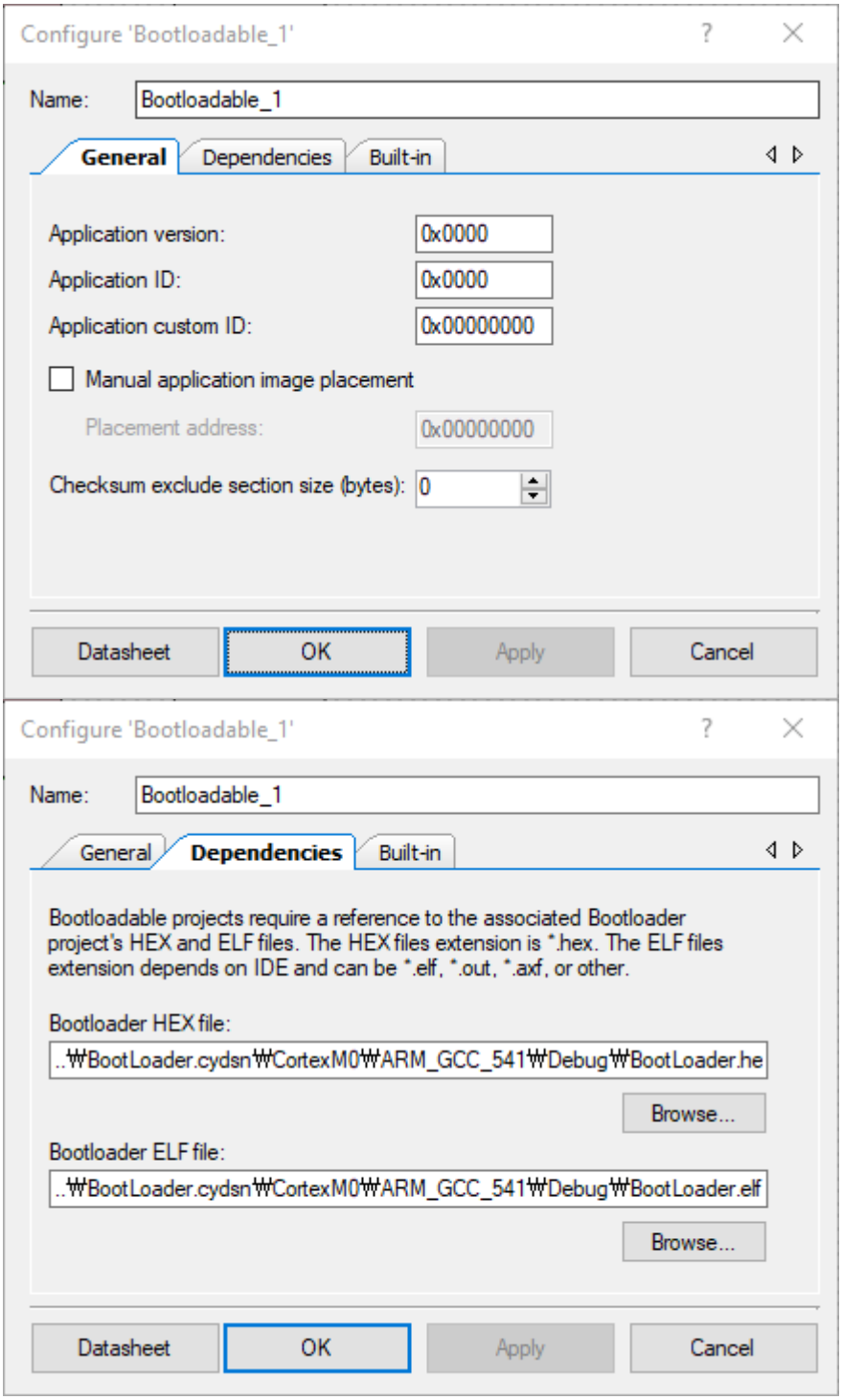
상기 Bootloader 프로젝트를 컴파일한 후, 디버거를 사용하여 해당 MCU 플래시에 프로그래밍하여 주세요.
필요시 해당 플래시 영역을 Protection 할 수 있습니다.
Bootloader 프로그램 프로그래밍은 easyDSP 로 수행될 수 없습니다.
한번 Bootloader 프로그램이 플래시에 프로그래밍된 이후에는 easyDSP 를 사용하여 Bootloadable 프로젝트를 플래시에 프로그래밍 할 수 있습니다.

STEP 3 : Bootloadable 프로젝트 작성

먼저 컴포넌트 카탈로그에서 하기와 같이 schematic 을 작성합니다.UART 컴포넌트의 이름을 UART_ezDSP 로 변경합니다.
기타 프로그램 동작을 위한 다른 컴포넌트는 여기에 표시하지 않았습니다.



각 컴포넌트의 설정은 하기와 같습니다.
먼저 Bootloader 컴포넌트 설정입니다. General 탭은 디폴트 그대로. Dependencies 탭에서 이전 Bootloader 프로젝트의 파일을 등록하세요.

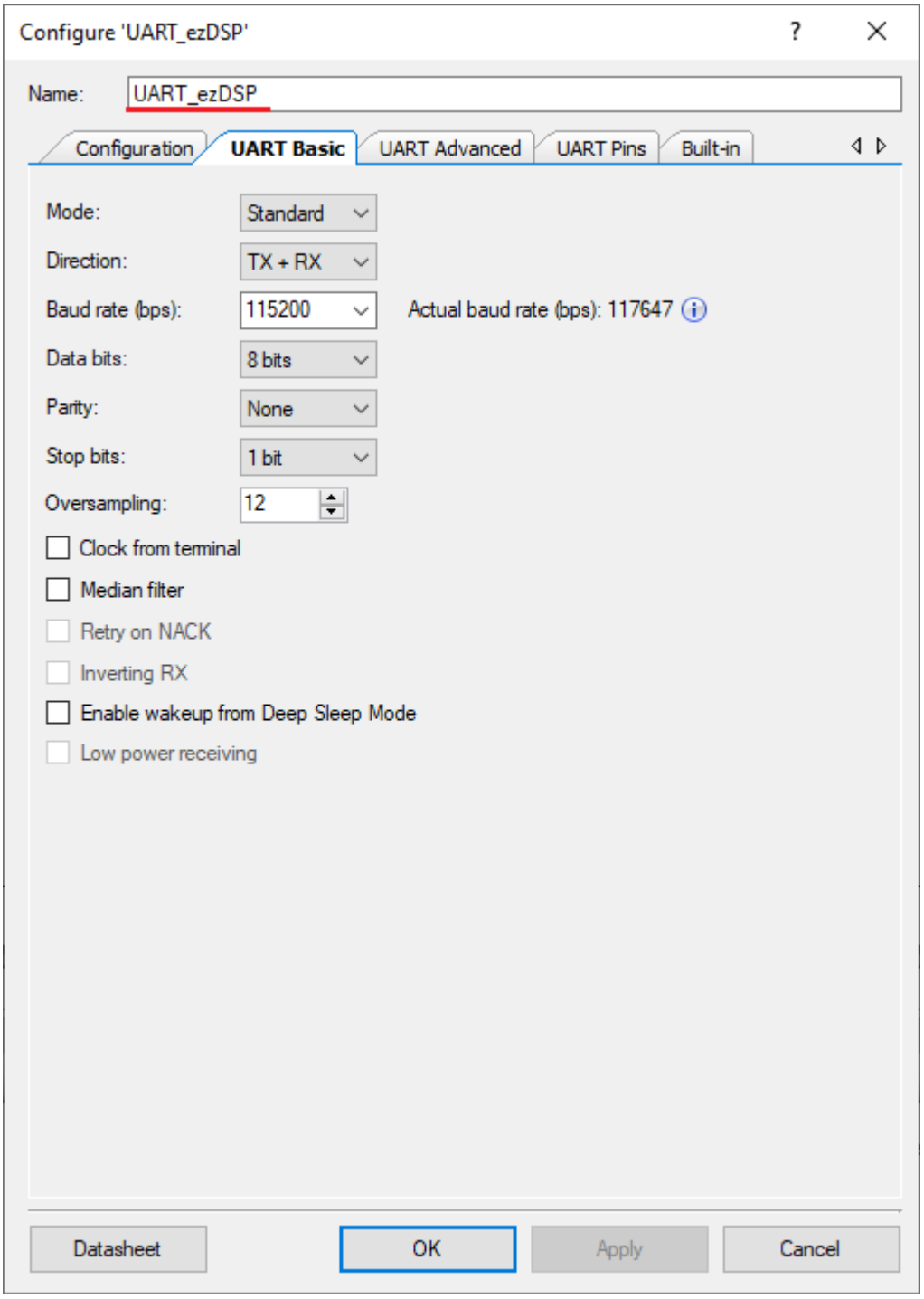


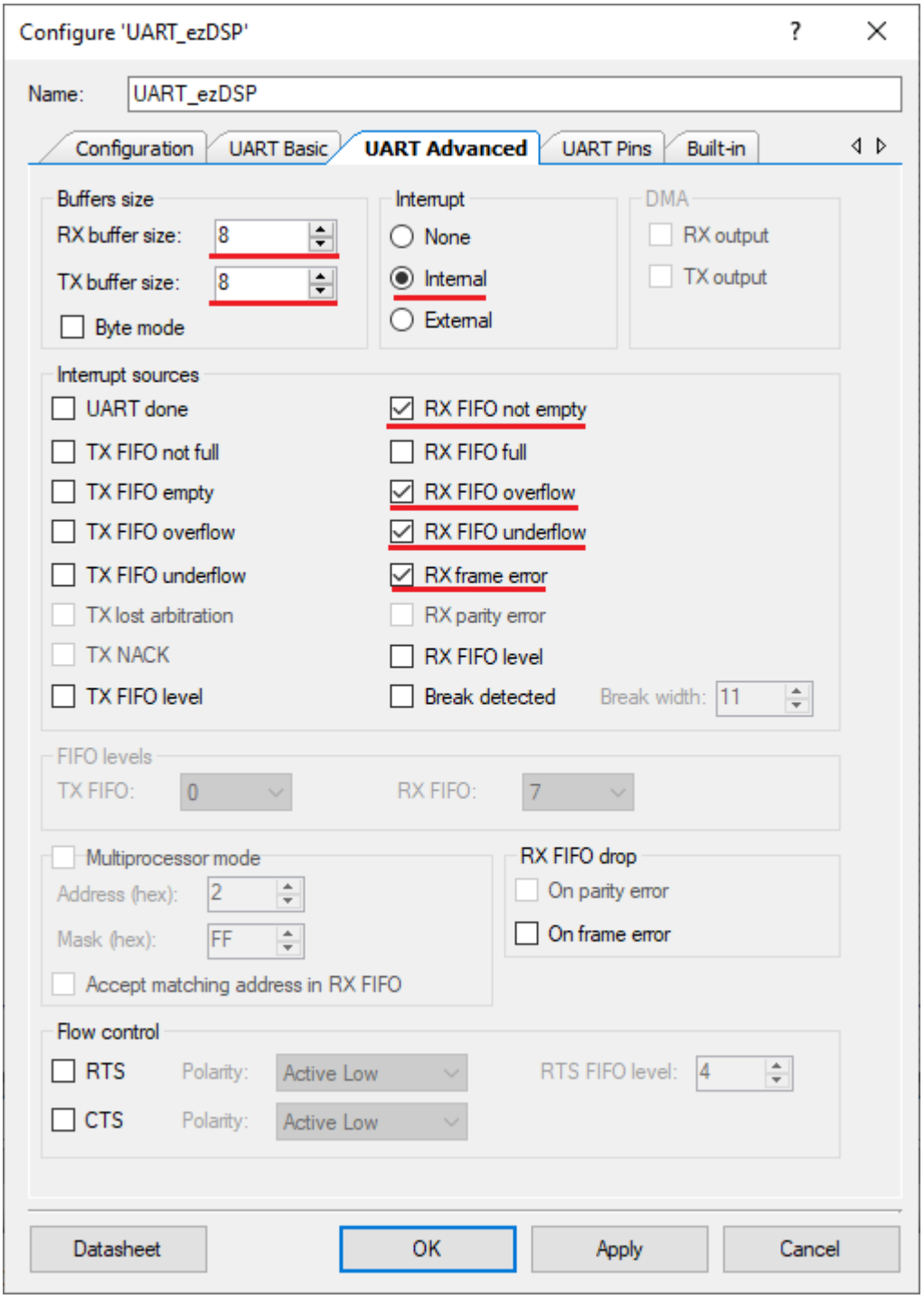
UART 컴포넌트 설정입니다.

UART Basic 탭에서는 easyDSP 통신 속도를 설정하시기 바랍니다. 부트로더의 통신 속도와 동일할 필요는 없습니다. 하지만 easyDSP 프로젝트에서 설정된 bps 와 동일해야 합니다.

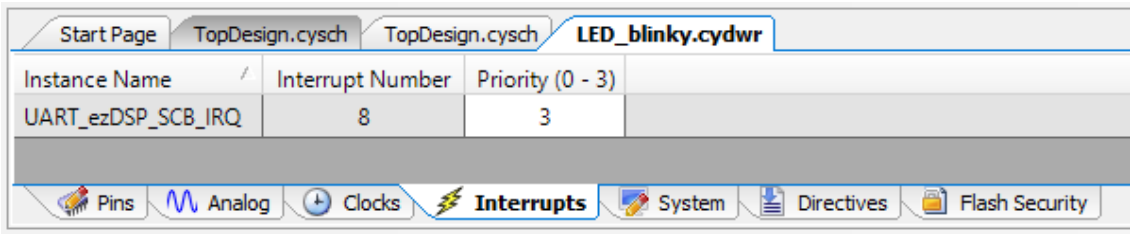
또한 8 비트, 패러티 없음, 1 스톱비트로 설정하세요.

UART Advanced 탭에서 하기 그림과 같이 설정해주세요.

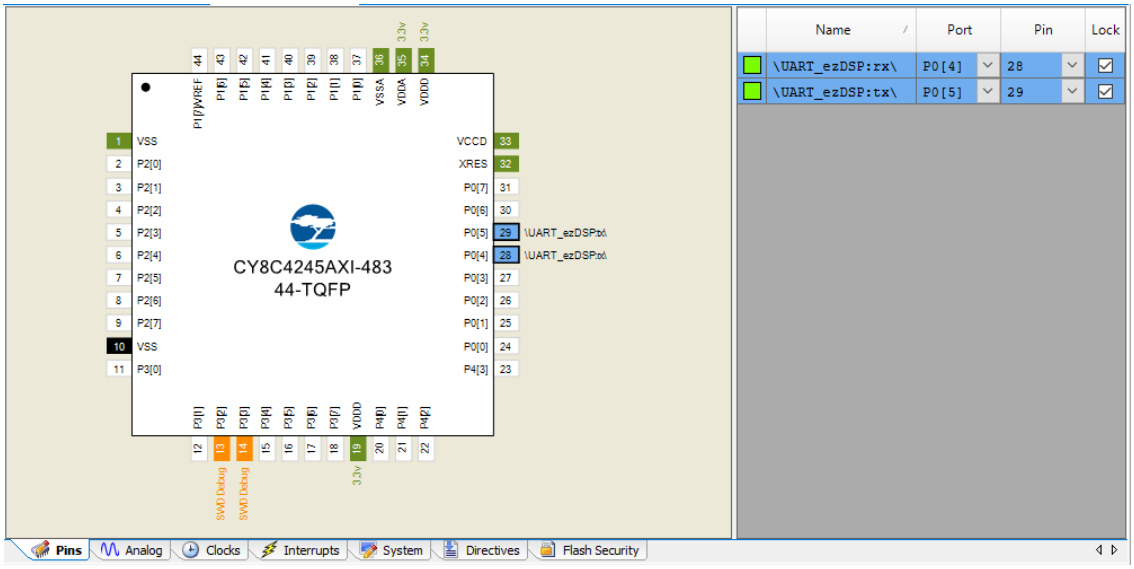

Configure 'UART_ezDSP'
Name:
UART_ezDSP
Configuration
UART Basic
UART Advanced
UART Pins
Built-in
Mode:
Standard
Direction:
TX + RX
Baud rate (bps):
115200
Actual baud rate (bps): 117647
Data bits:
8 bits
Parity:
None
Stop bits:
1 bit
Oversampling:
12
Clock from terminal
Median filter
Retry on NACK
Inverting RX
Enable wakeup from Deep Sleep Mode
Low power receiving
Datasheet
OK
Apply
Cancel



easyDSP 를 위한 UART 인터럽트 순위는 최하위로 설정 바랍니다.



UART 핀은 사용자 보드에 맞게 선정하시기 바랍니다. 앞서 부트로더 프로젝트와 동일한 핀 선정이 되어야 합니다. 본 예제에서는 P0.4, P0.5 에 할당되었습니다.



STEP 4 : easyDSP_init() 함수 호출

easyDSP 통신을 위해 제공되는 소스파일(easyPSoC4.h, easyPSoC4.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩PSoC 에서 찾을 수 있습니다.

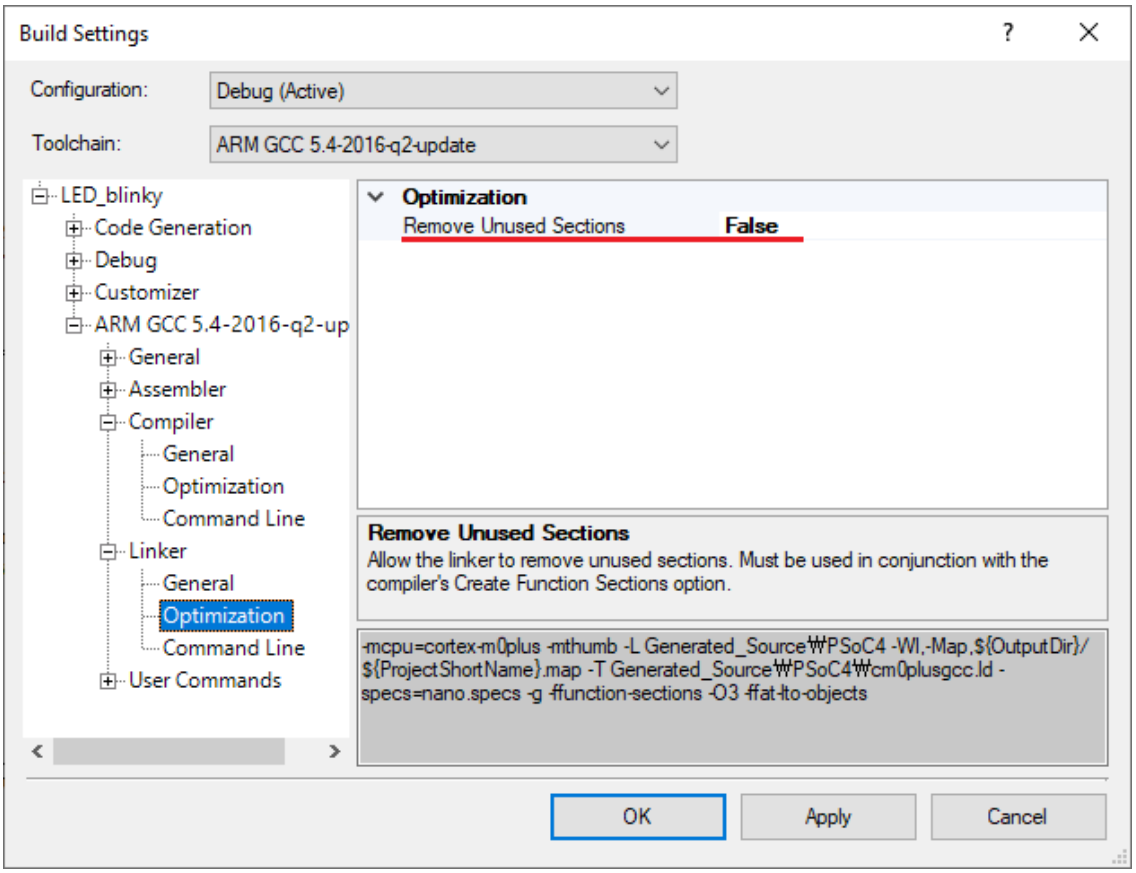
마지막으로 main() 함수에서 easyDSP_init() 함수를 호출하시기 바랍니다.
이로서 easyDSP 사용을 위한 부트로더블 프로젝트가 완료되었습니다.

```
#include "project.h"
#include "easyPSoC4.h"

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g. MyInst_Start()) */
    easyDSP_init();

    for(;;)
    {
        /* Place your application code here. */
    }
}
```

STEP 5 : IDE 설정

1. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.
2. 매 컴파일마다 cyacd 파일이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 해주세요. cyacd 파일은 플래시 프로그래밍할 때 사용됩니다.
3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아

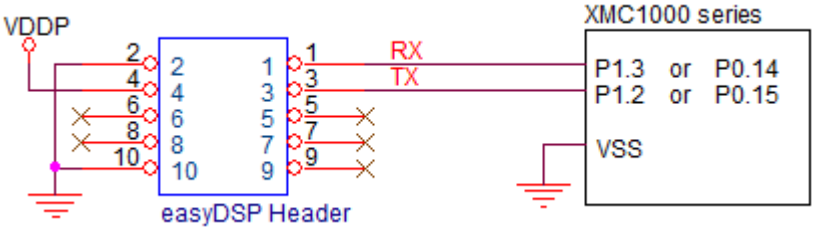
easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수를 포함되게 하기 위해서라면
예를 들어 PSoC Creator 4.4 에서 링커 옵션에 'Remove Unused Sections' 항목을 False 로 설정하시기 바랍니다.
다른 개발 환경이라면 적절히 해당하는 세팅이 필요합니다.



7.8 XMC1

STEP 1 : 하드웨어 설정

UART 핀 (P1.3/P1.2 조합 또는 P0.14/P0.15 조합)을 easyDSP RX/TX 핀과 직접 연결합니다 .
RX, TX 신호는 easyDSP 포드 내에서 100k 오옴으로 풀업되어 있습니다.
easyDSP 헤더 4 번핀에는 MCU 의 VDDP 를 연결하여 주십시요 .



STEP 2 : easyDSP 제공 헤더 파일 수정

먼저 easyDSP 통신을 위해 제공되는 소스파일 (easyXMC1.h, easyXMC1.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩XMC 에서 찾을 수 있습니다.
본 소스파일에서 XMC Peripheral Library 를 사용하고 있으므로 해당 라이브러리도 프로젝트에 포함되어 있어야 합니다.

easyXMC1.h 파일에서 통신 채널, 보드레이트를 설정하시기 바랍니다.
easyDSP 프로젝트에서 사용될 보드레이트와 동일하게 설정합니다.

총 64 개의 FIFO 버퍼 중 easyDSP 용으로 입력/출력 각 8 개를 선정합니다. easyDSP 사용 USIC 모듈의 다른 채널 FIFO 버퍼와의 충돌을 회피해야 합니다.

```
////////////////////////////////////////////////
// Select channel (0 or 1) of USIC0 :
// define 1 to target channel. 0 to another.
////////////////////////////////////////////////
#define USE_USIC0_CH0       0       // use USIC0 channel 0 with pins P0.14 + P0.15
#define USE_USIC0_CH1       1       // use USIC0 channel 1 with pins P1.3 + P1.2

////////////////////////////////////////////////
// Set UART configuration
////////////////////////////////////////////////
#define EZ_BUAD_RATE        230400U     // baud rate for UART communication for easyDSP
#define EZ_TX_BUFFER_START  0           // FIFO Buffer Partitioning for channel 0 of USIC0.
#define EZ_RX_BUFFER_START  8           // FIFO Buffer Partitioning for channel 0 of USIC0.
```

STEP 3 : easyDSP_init() 함수 호출

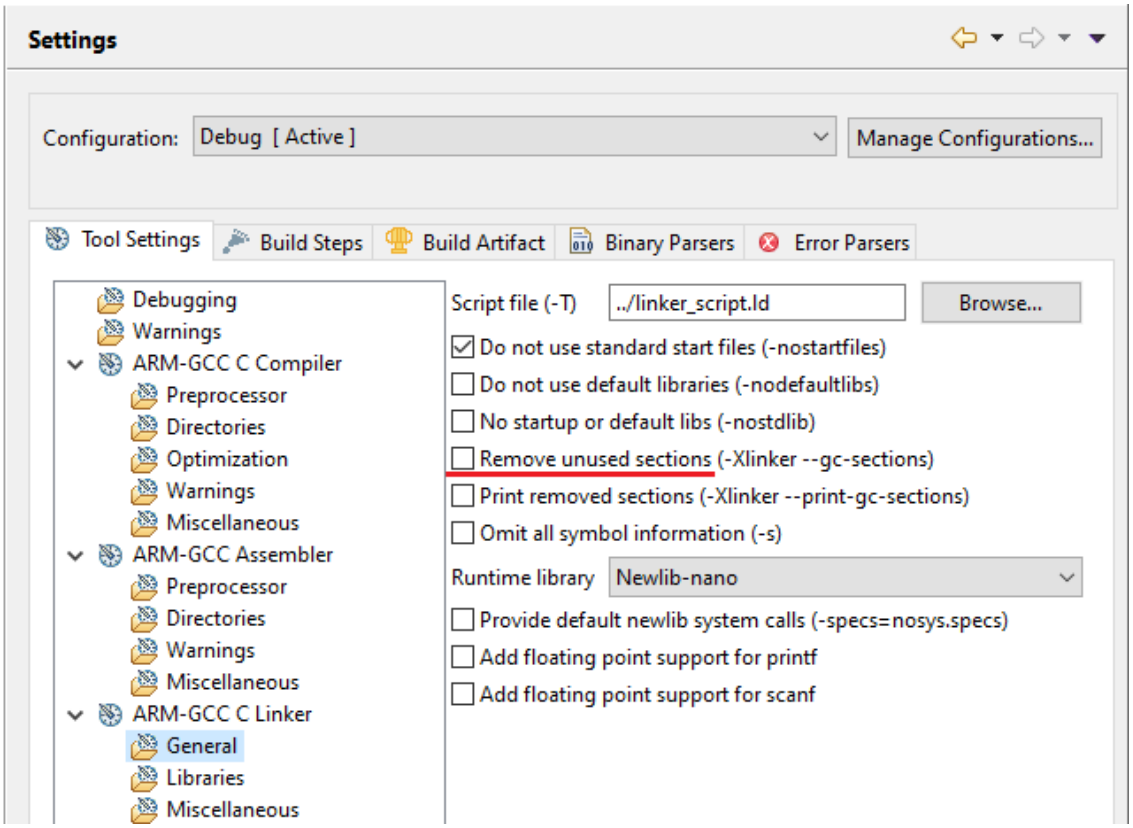
먼저 main.c 상단에 easyXMC1.h 를 include 하여 주시고,
main 함수 적절 부분에 easyDSP_init() 함수를 호출하시기 바랍니다.

```
#include "easyXMC1.h"

int main(void)
{




    easyDSP_init();

    while(1);
}
```

STEP 4 : IDE 설정

1. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

2. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수를 포함되게 하기 위해서라면
예를 들어 Dave 에서 링커 옵션에 'Remove Unused Sections' 항목을 클릭하시지 마세요.
다른 개발 환경이라면 적절히 해당하는 세팅이 필요합니다.



7.9 XMC4

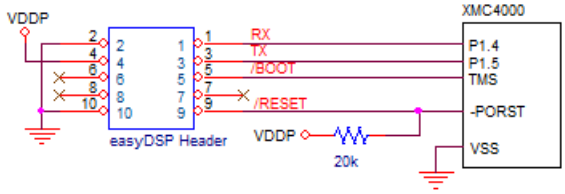
STEP 1 : 하드웨어 설정

XMC4 는 파워온 리셋 이후 TCK, TMS 핀 상태에 따라 4 가지 부트 모드를 지원합니다.

TCK	TMS	Boot mode
0	1	Normal
0	0	ASC BSL
1	1	BMI
1	0	CAN BSL

easyDSP 는 이 중 Normal 모드 (플래시로 부팅) 또는 ASC BSL 모드 (UART 채널로 부팅하여 플래시 프로그래밍)만을 지원합니다.
따라서 파워온 리셋시 TCK 핀은 Low 상태를, TMS 핀 상태는 easyDSP 가 선택할 수 있도록 결선되어야 합니다.

MCU 내부에서 TCK 핀은 weak pull-down 되어 있으며, TMS 핀은 weak pull-up 되어 있으므로, 해당핀의 외부 풀다운/풀업 저항은 옵션사항입니다.
UART 핀 P1.4, P1.5 을 easyDSP RX/TX 핀과 직접 연결합니다 . RX, TX 신호는 easyDSP 포드 내에서 100k 오옴으로 풀업되어 있습니다.
easyDSP 헤더 4 번핀에는 MCU 의 VDDP 를 연결하여 주십시요 .



기타 주의 사항 :
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU -PORST 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 -PORST 에 전달해야 함
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함

STEP 2 : easyDSP 제공 헤더 파일 수정

먼저 easyDSP 통신을 위해 제공되는 소스파일 (easyXMC4.h, easyXMC4.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩XMC 에서 찾을 수 있습니다.
본 소스파일에서 XMC Peripheral Library 를 사용하고 있으므로 해당 라이브러리도 프로젝트에 포함되어 있어야 합니다.

easyXMC4.h 파일에서 easyDSP 통신 보드레이트를 설정하시기 바랍니다.
easyDSP 프로젝트에서 사용될 보드레이트와 동일하게 설정합니다.

총 64 개의 FIFO 버퍼 중 easyDSP 용으로 입력/출력 각 8 개를 선정합니다. easyDSP 사용 USIC 모듈의 다른 채널 FIFO 버퍼와의 충돌을 회피해야 합니다.

```
////////////////////////////////////////////////
// Select other configuration
////////////////////////////////////////////////

#define EZ_BUAD_RATE        230400U     // baud rate for UART communication for easyDSP
#define EZ_TX_BUFFER_START  0           // FIFO Buffer Partitioning for channel 0 of USIC0.
#define EZ_RX_BUFFER_START  8           // FIFO Buffer Partitioning for channel 0 of USIC0.
```

STEP 3 : easyDSP_init() 함수 호출

먼저 main.c 상단에 easyXMC4.h 를 include 하여 주시고,
main 함수 적절 부분에 easyDSP_init() 함수를 호출하시기 바랍니다.

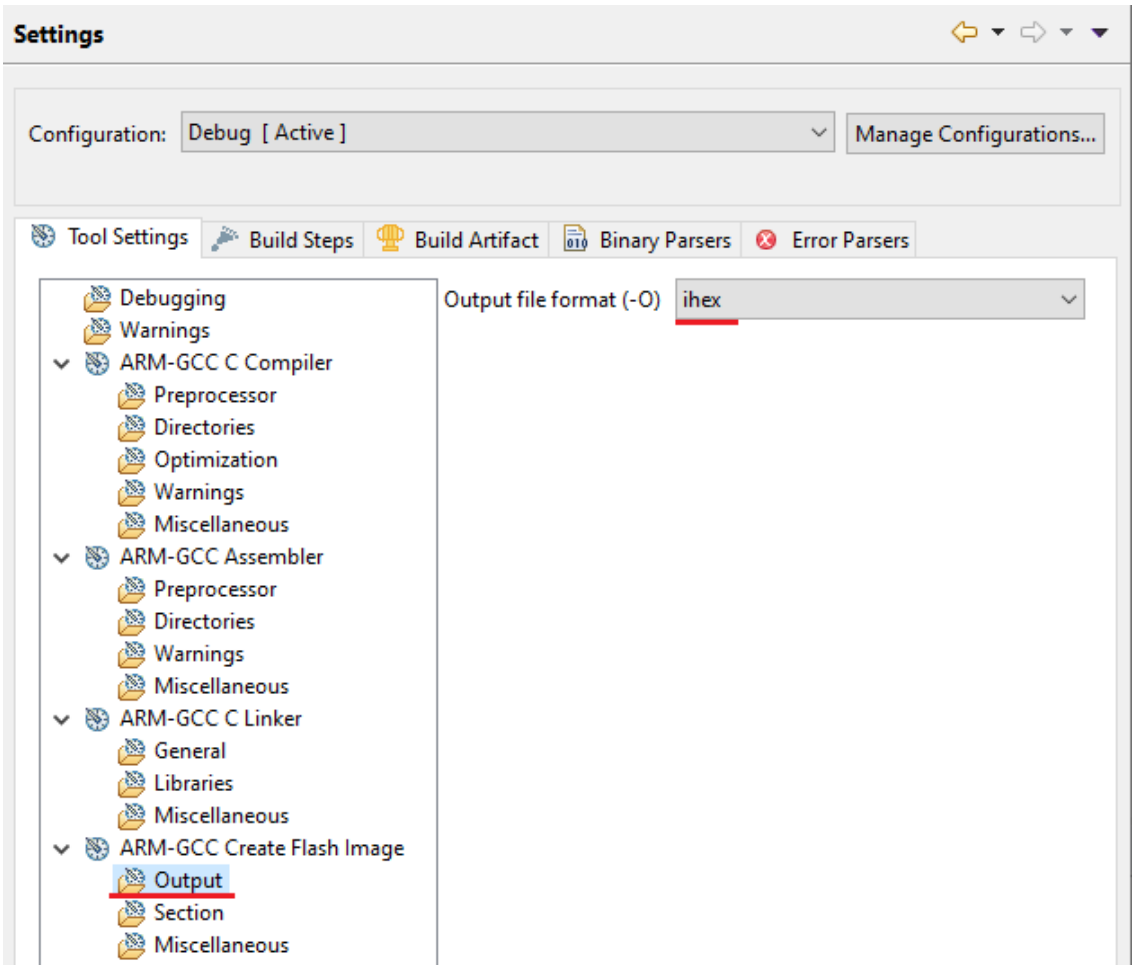
```
#include "easyXMC4.h"

int main(void) {



    easyDSP_init();

    /* Infinite loop */
    for(;;) {
    }
}
```

STEP 4 : IDE 설정

1. 매 컴파일마다 hex 파일 (인텔 형식) 이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 IDE 를 설정해주세요. Hex 파일은 플래시 프로그래밍할 때 사용됩니다.

Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.

예를 들어 DAVE IDE 의 경우 :

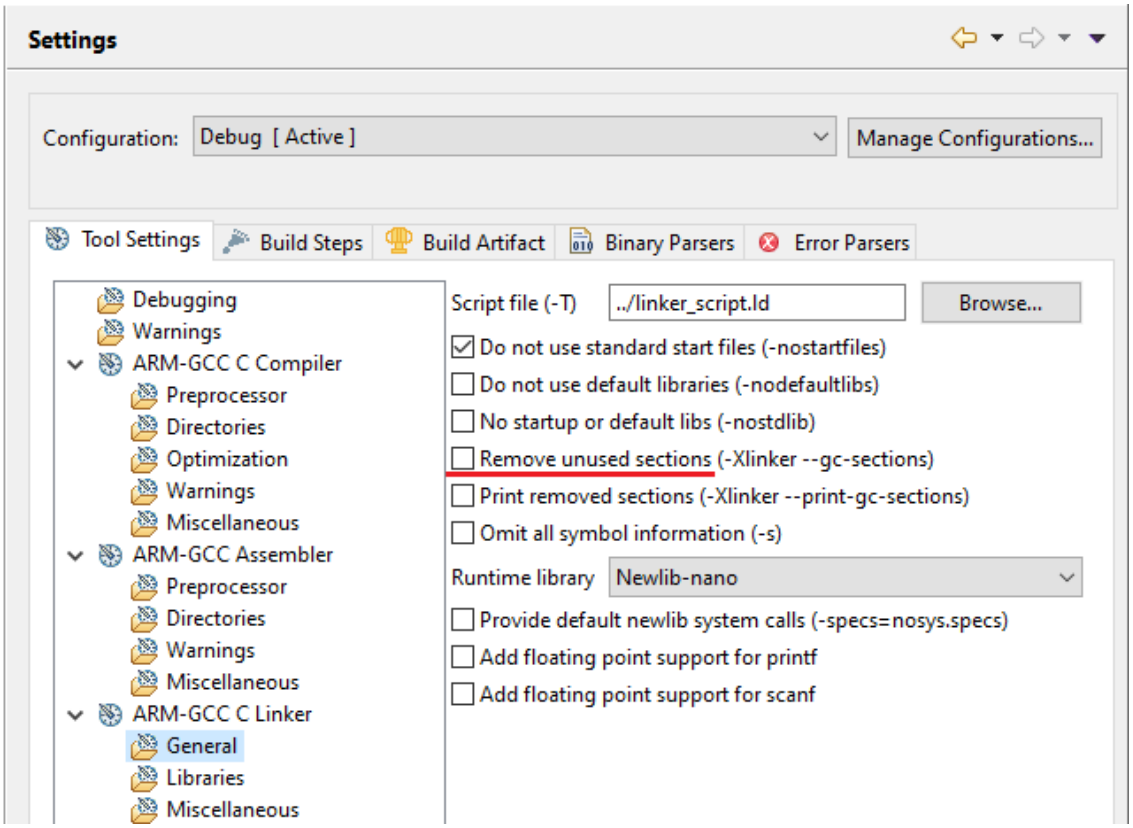


2. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수를 포함되게 하기 위해서라면

예를 들어 Dave 에서 링커 옵션에 'Remove Unused Sections' 항목을 클릭하시지 마세요.

다른 개발 환경이라면 적절히 해당하는 세팅이 필요합니다.

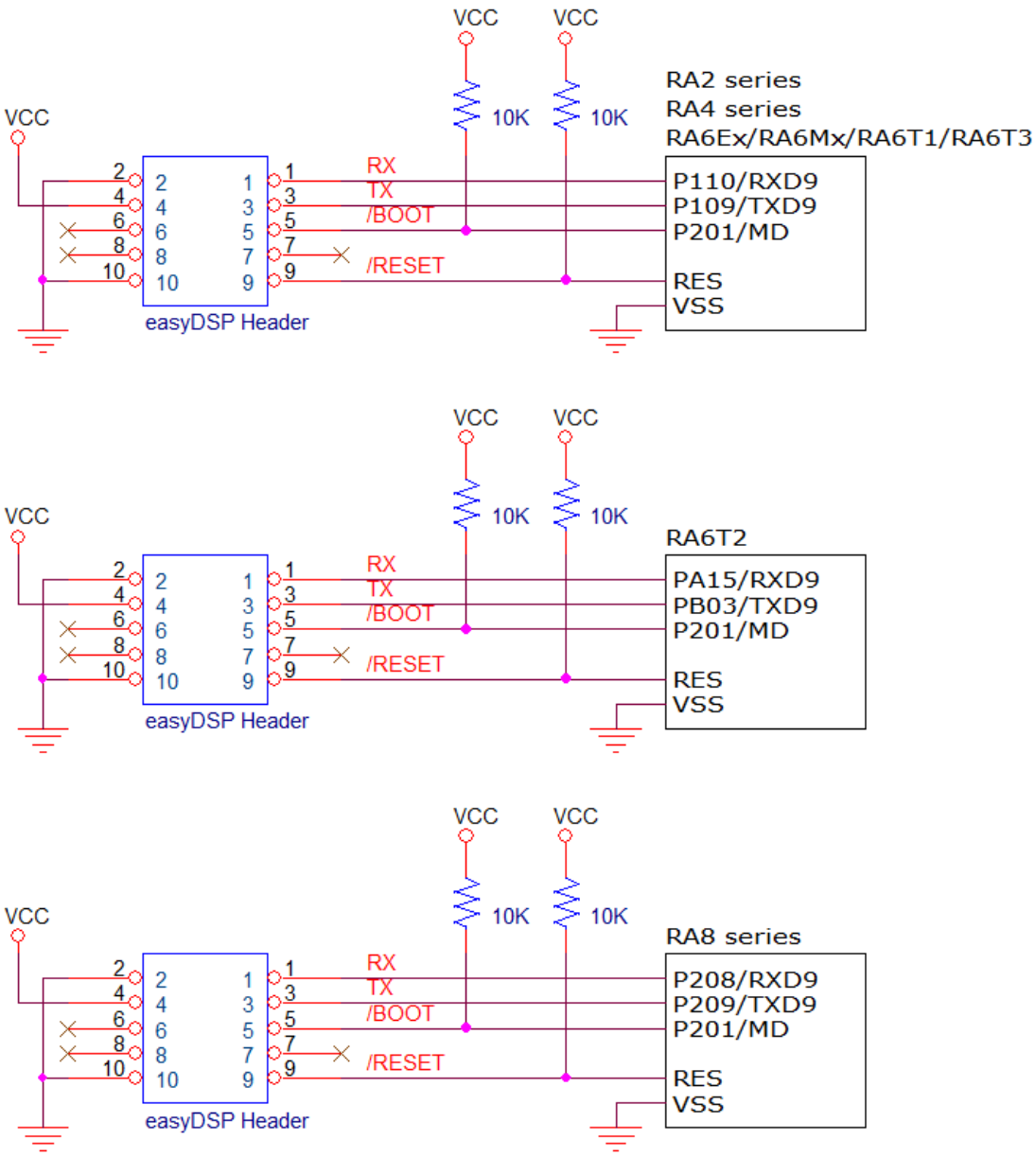


7.10 RA

7.10.1 RA 하드웨어 설정

easyDSP 와 연결

SCI9 을 사용해야 easyDSP 가 플래시 프로그래밍을 지원할 수 있습니다.
따라서 하기 결선과 같이 SCI9 의 RXD9, TXD9 을 easyDSP 해당핀과 직결 연결 권장 드립니다.
easyDSP 헤더 RX, TX 신호는 내부에서 100k 오옴으로 풀업되어 있습니다.
easyDSP 헤더 4 번 핀에는 MCU 의 VCC 를 연결하여 주십시요.



기타 주의 사항 :

- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU RES 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 RES 에 전달해야 함
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함

만약 SCI9 을 사용하실 수 없는 상황이라면, 다른 SCI 핀을 연결할 수 있습니다.
하지만 이 경우 플래시 프로그래밍이 지원되지 않고, 변수 모니터링만 지원되며, /BOOT 핀과 /RESET 핀 연결은 필요하지 않습니다.

디버거와의 호환성

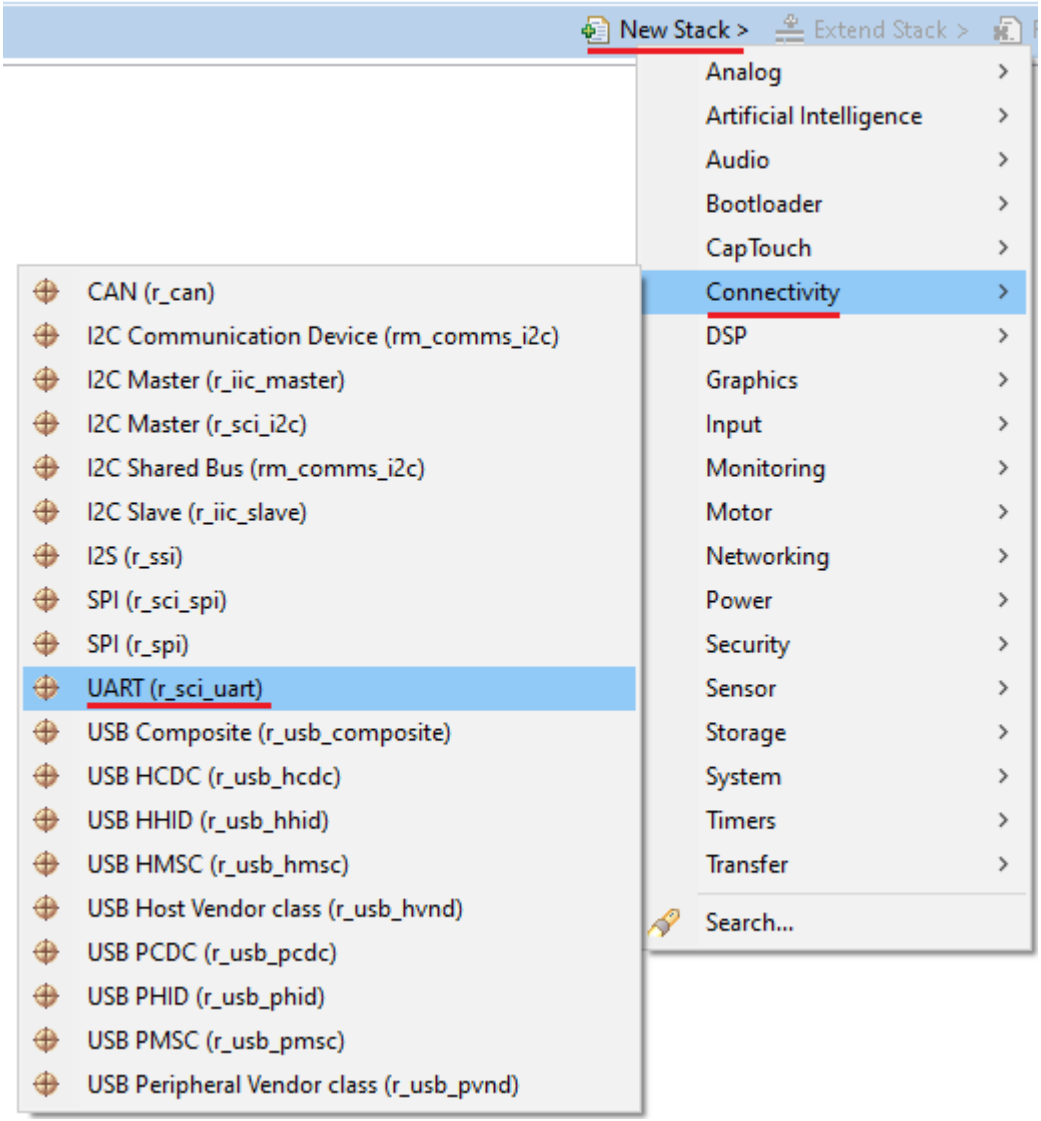
easyDSP 는 RXD9, TXD9 핀을 통해 MCU 를 모니터링합니다. 해당 핀은 일부 RA MCU 에서 디버거 핀과 중복됩니다 (JTAG TDO, TDI 핀 및 SWD SWO 핀). RA2 계열 MCU 에는 해당 디버거 핀이 없으므로 문제 없습니다만, RA4, RA6, RA8 계열 일부 MCU 에서는 해당 디버거 핀 사용이 불가능해지므로 easyDSP 와 같이 디버거를 사용하려면, SWO 없는 SWD 를 사용해야 합니다.

7.10.2 RA 소프트웨어 설정

easyDSP 는 FSP(Flexible Software Package)에서 생성된 소스코드를 통신 채널 설정에 사용하고 있습니다.
하기에서는 FSP v3.5.0 기준으로 설명드립니다.

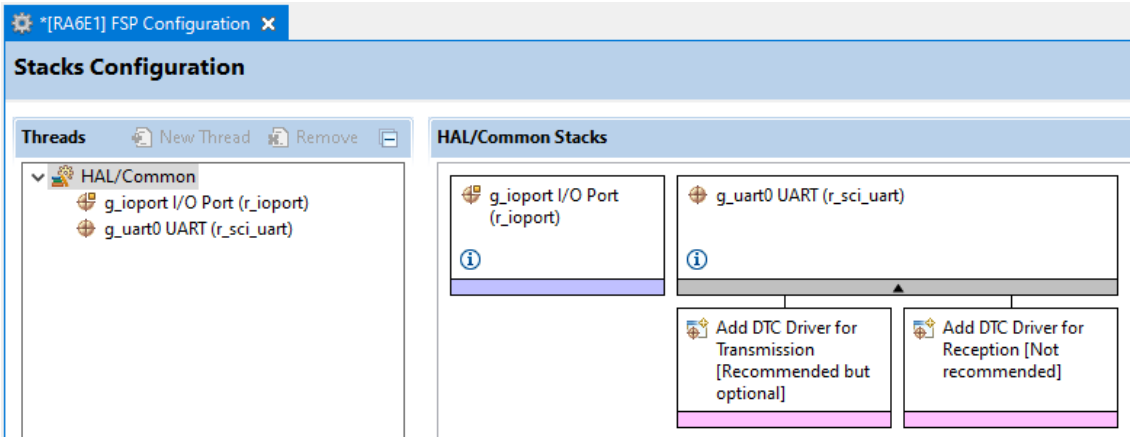
STEP 1 : FSP(Flexible Software Package) 설정

configuration.xml 파일을 클릭하여 FSP 를 호출합니다.
Stacks 탭에서 먼저 UART 스택을 생성합니다. MCU 종류에 따라 r_sci_uart 모듈 또는 r_sci_b_uart 모듈을 사용합니다.

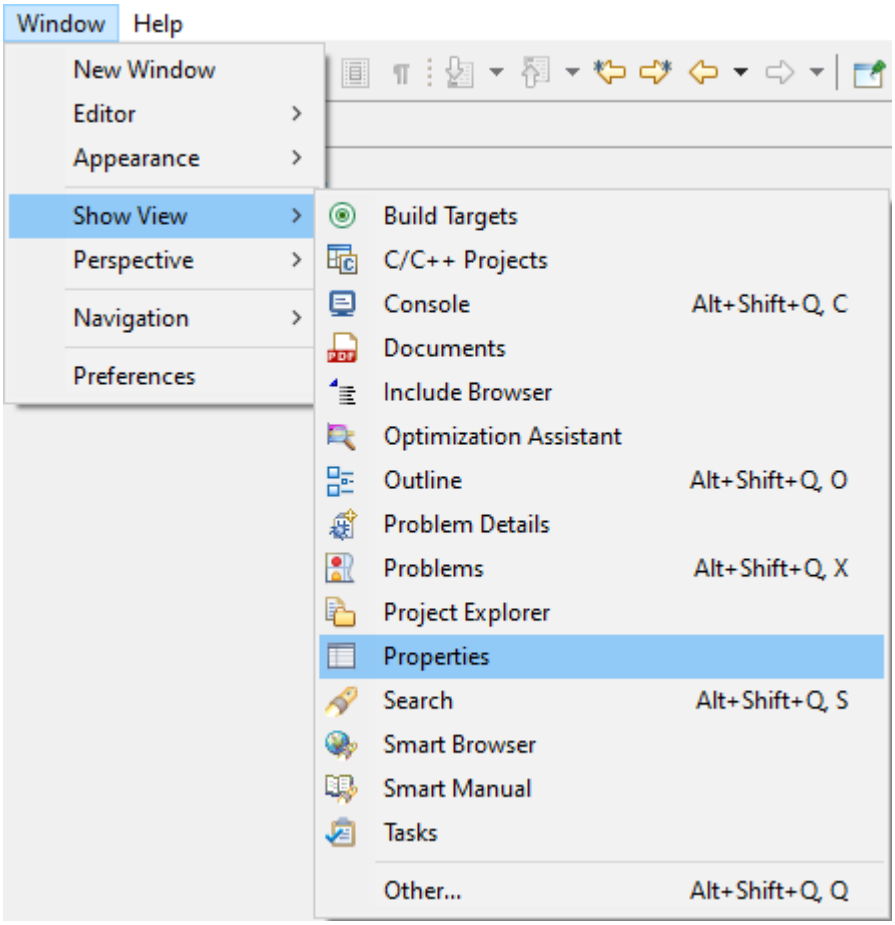


DTC Driver 는 사용하지 않으므로 해당 기능은 수정하지 않습니다.

생성된 UART 스택을 클릭하여 Properties 창에서 상세 내용을 설정합니다.



아래 Properties 창이 보이지 않을 경우, 하기와 같이 활성화시켜 주세요.



하기 빨간색이 수정 내용입니다.

MCU 에 따라 FIFO 가 지원되는 채널이 다르지만, UART 채널 9 에 FIFO 가 지원된다면 사용합니다. NEW!

먼저 모듈 이름을 g_easyDSP 로 변경합니다.

통신 채널은 SCI9 를 사용하므로 9 로 변경하며, 보드레이트는 적절히 선택합니다 (추후 easyDSP 프로젝트 설정과 동일해야 함).

인터럽트 콜백의 이름은 easyDSP_callback 으로 변경하고, 인터럽트 순위는 최하순위로 설정합니다.

easyDSP 가 MCU 의 부트모드를 이용하여 플래시 프로그래밍하기 위해서는 SCI9 채널의 TXD9, RXD9 을 사용해야 합니다.

MCU 종류에 따라 해당 핀이 달라지지만 (RA 하드웨어 설정 참조), 본 설명에서는 P109, P110 인 경우를 산정합니다.

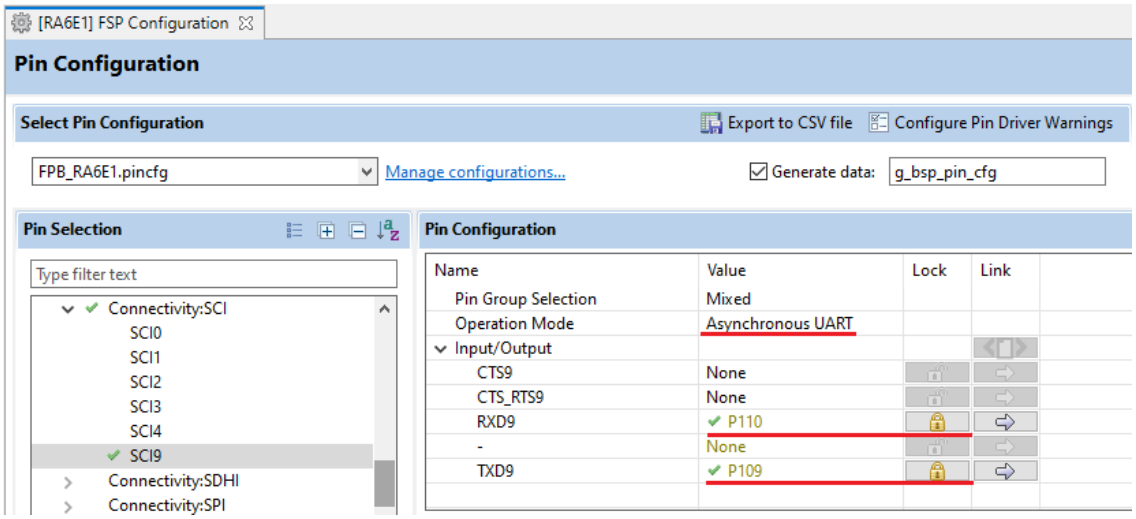
Properties | Problems | Console | Smart Browser | Smart Manual

g_easyDSP UART (r_sci_uart)

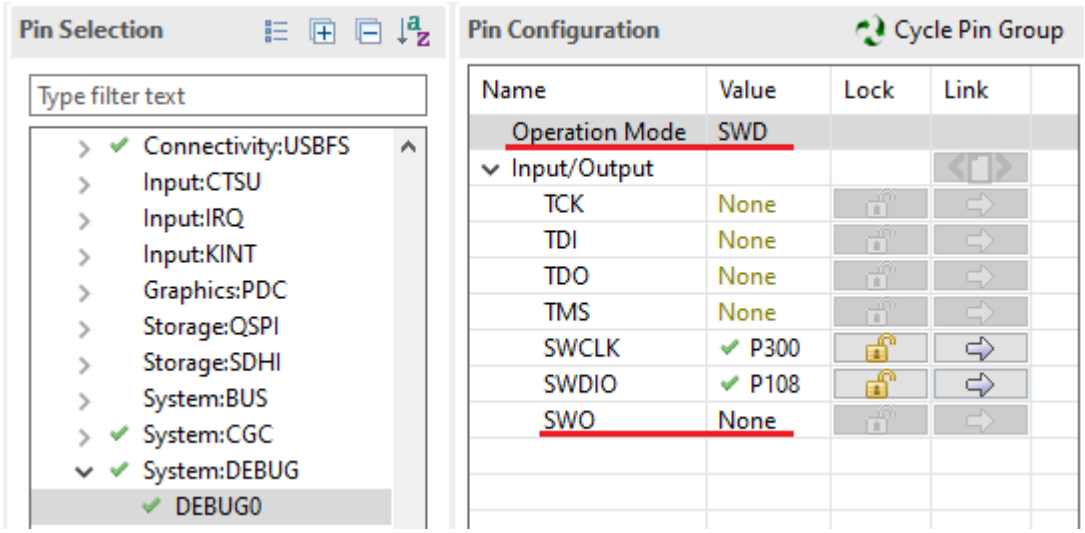
Settings | API Info

Property	Value
Common	
Parameter Checking	Default (BSP)
FIFO Support	Enable
DTC Support	Disable
Flow Control Support	Disable
RS-485 Support	Disable
Module g_easyDSP UART (r_sci_uart)	
General	
Name	g_easyDSP
Channel	9
Data Bits	8bits
Parity	None
Stop Bits	1bit
Baud	
Baud Rate	115200
Baud Rate Modulation	Disabled
Max Error (%)	5
Flow Control	
CTS/RTS Selection	Hardware RTS
Software RTS Port	Disabled
Software RTS Pin	Disabled
Extra	
RS-485	
DE Pin	Disable
DE Pin Polarity	Active High
DE Port Number	Disabled
DE Pin Number	Disabled
Clock Source	Internal Clock
Start bit detection	Falling Edge
Noise Filter	Disable
Receive FIFO Trigger Level	One
Interrupts	
Callback	easyDSP_callback
Receive Interrupt Priority	Priority 15
Transmit Data Empty Interrupt Priority	Priority 15
Transmit End Interrupt Priority	Priority 15
Error Interrupt Priority	Priority 15
Pins	
CTS9	None
CTS_RTS9	None
RXD9	P110
TXD9	P109

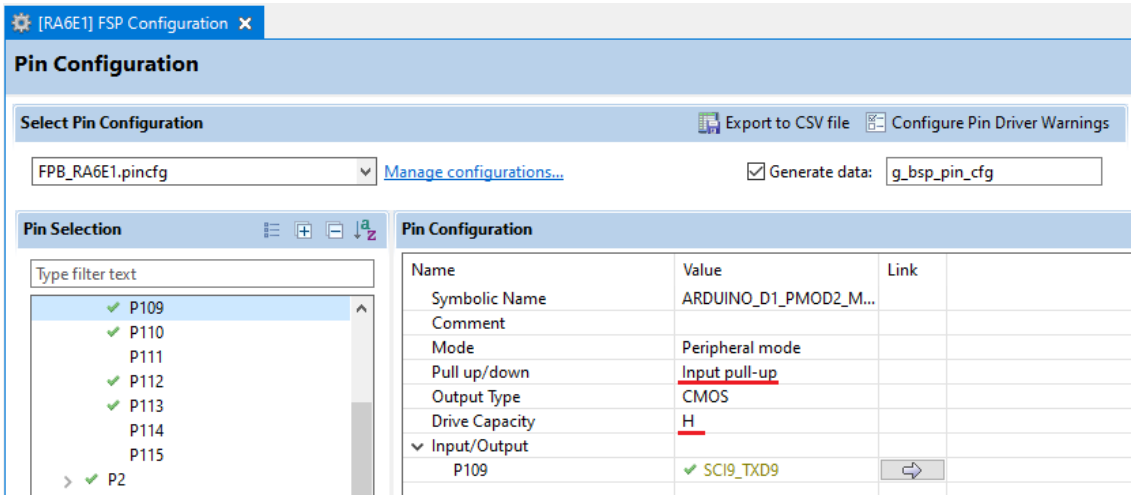
Pins 탭으로 이동하여 SCI9 의 Operation Mode 가 Asyncronous UART 가 되도록 하여 주시고, TXD9, RXD9 에 각각 P109, P110 핀을 지정합니다.

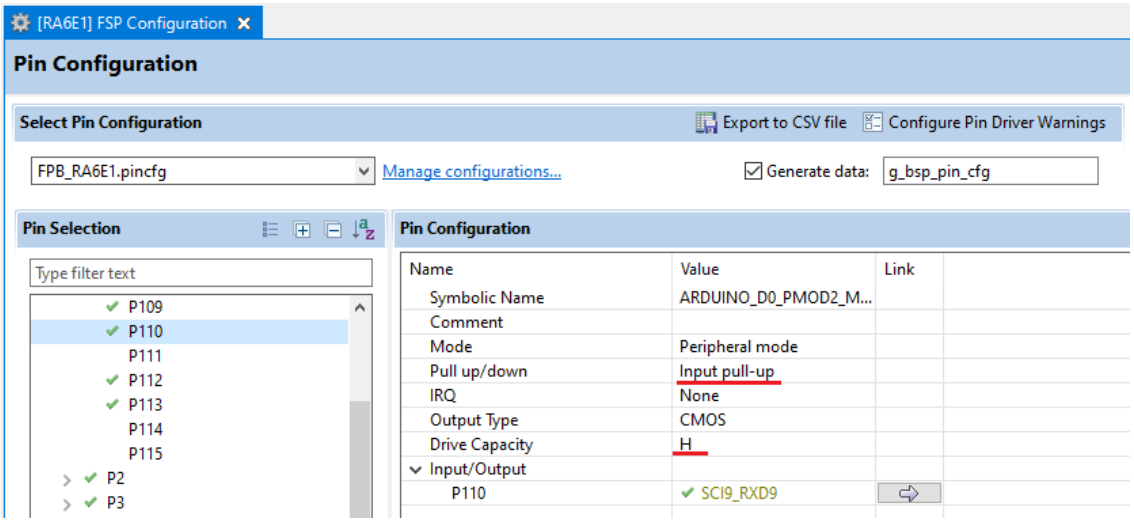


RA4, RA6, RA8 계열 일부 MCU 경우, 상기 RXD9, TXD9 에 할당된 핀이 JTAG 에 할당될 경우 핀 중첩이 발생하게 되므로, 디버거로는 SWD (SWO 미사용)를 사용하여 주십시요.

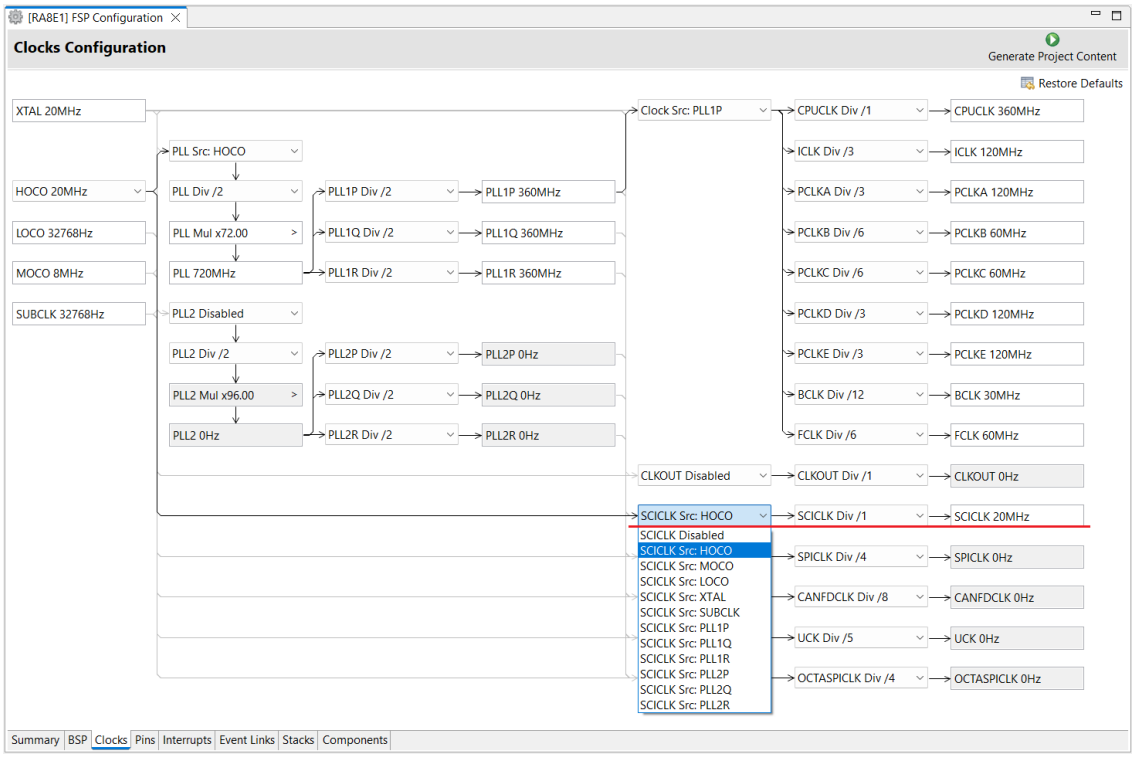


또한 RXD9, TXD9 핀에 대해서 내부 풀업 및 전류 용량을 증대시키시기 바랍니다.





일부 MCU 의 경우 (예:RA8E1) SCI 에 사용되는 클럭을 활성화/비활성화시킬 수 있으며, 이 경우 반드시 활성화시켜야 합니다.



마지막으로 코드를 생성합니다.

STEP 2 : easyDSP_init() 함수 호출

먼저 easyDSP 통신을 위해 제공되는 소스파일 (easyRA_v11.4.h, easyRA_v11.4.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩RA 에서 찾을 수 있습니다.
마지막으로 hal_entry() 함수에서 easyDSP_init() 함수를 호출하시기 바랍니다.

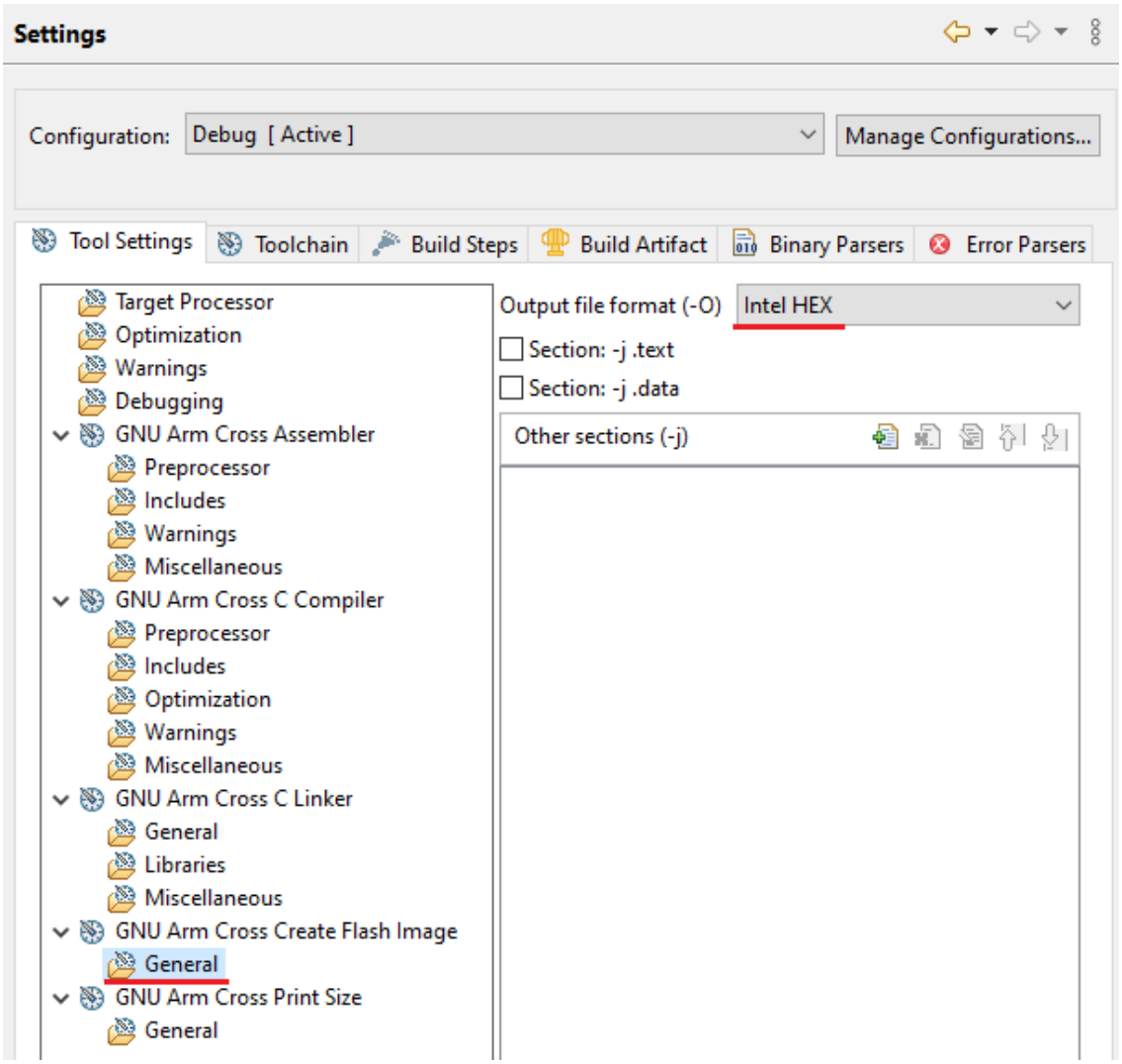
```
#include "easyRA_v11.4.h"

void hal_entry(void)
{

    .
    .
    .
    .
    .
    .

    easyDSP_init();
}
```

STEP 3 : IDE 설정

1. 매 컴파일마다 hex 파일(인텔 형식)이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 개발 환경을 설정 해주세요. hex 파일은 플래시 프로그래밍할 때 사용됩니다.

Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.

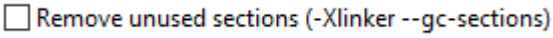
예를 들어 e2 stuio 경우 :



2. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수가 포함되게 하기 위해서라면

예를 들어 e2Studio 에서 링커 옵션에 'Remove Unused Sections' 항목을 클릭하시지 마세요. 다른 개발 환경이라면 적절히 해당하는 세팅이 필요합니다.

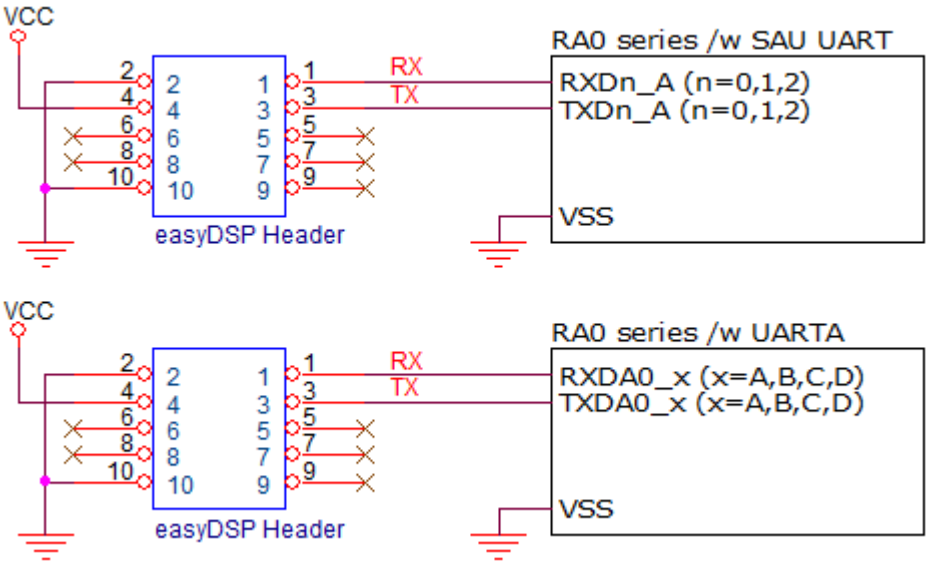


7.10.3 RA0 설정

easyDSP 와 연결

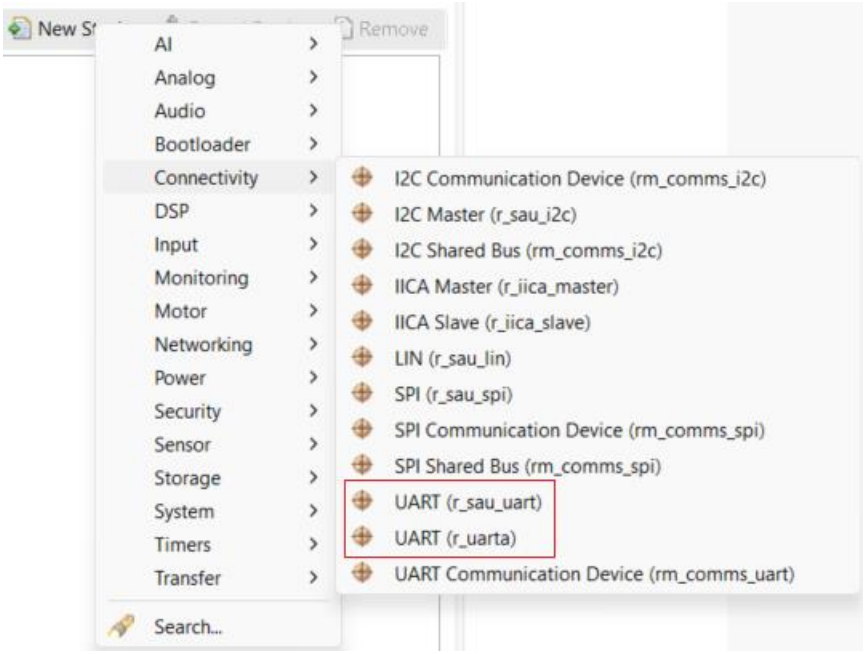
RA0 의 경우 플래시 프로그래밍을 지원하지 않으므로 easyDSP 포드의 TX, RX 만 MCU 와 연결합니다.
MCU 의 SAU 또는 UARTA 중에 사용자 보드 상황에 맞게 선택 사용할 수 있습니다.

easyDSP 헤더 RX, TX 신호는 포드 내부에서 100k 오옴으로 풀업되어 있습니다.

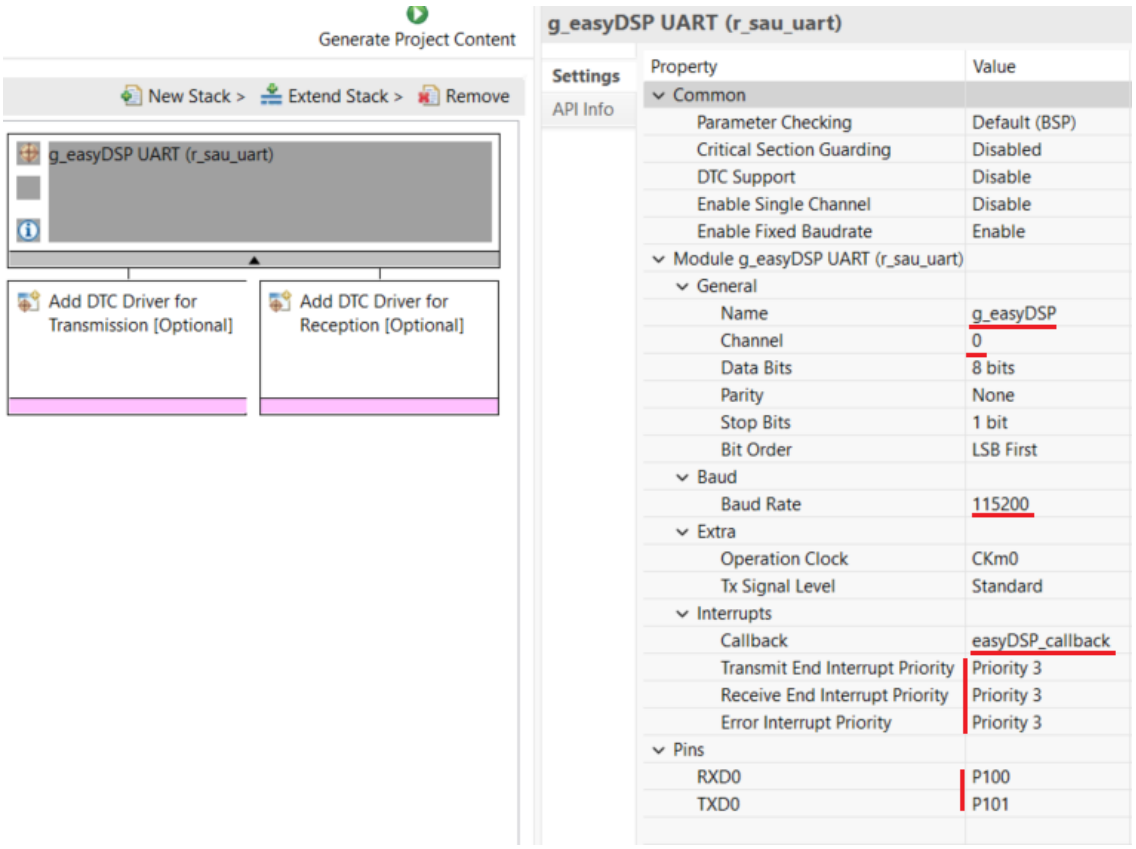


FSP (Flexible Software Package) 설정

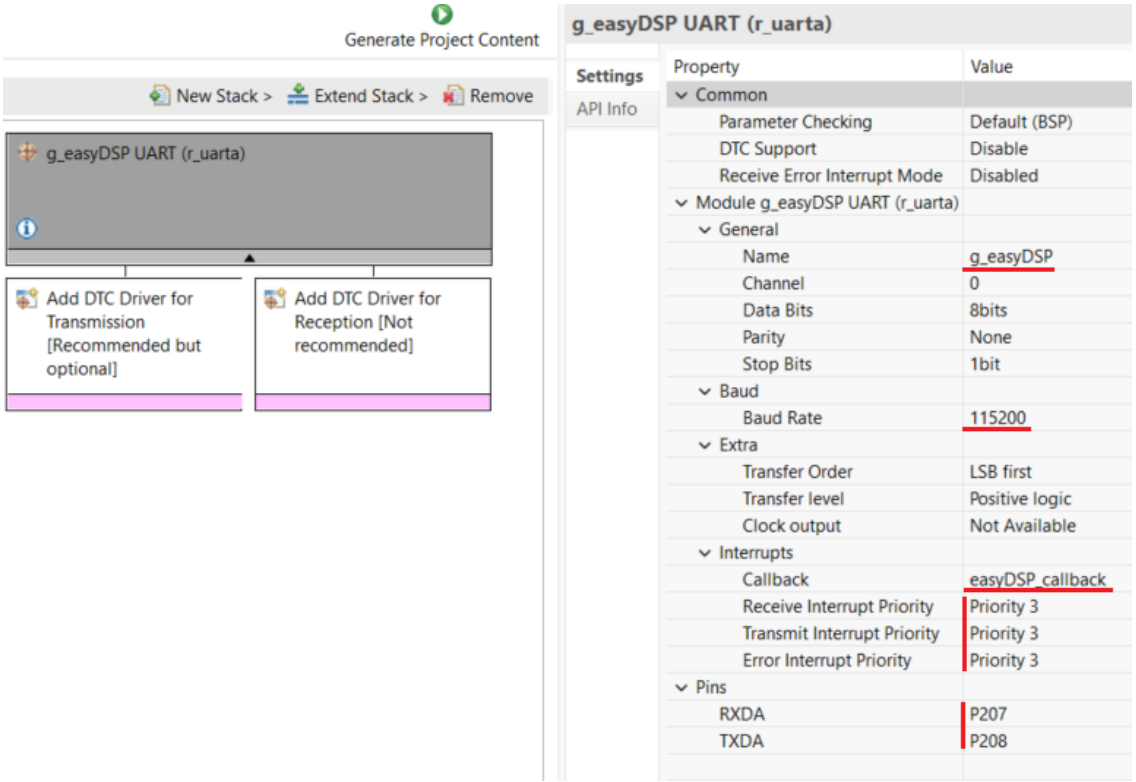
configuration.xml 파일을 클릭하여 FSP 를 호출합니다.
먼저 Stacks 탭에서 UART 스택을 생성합니다. r_sau_uart 또는 r_uarta 모듈을 사용합니다.



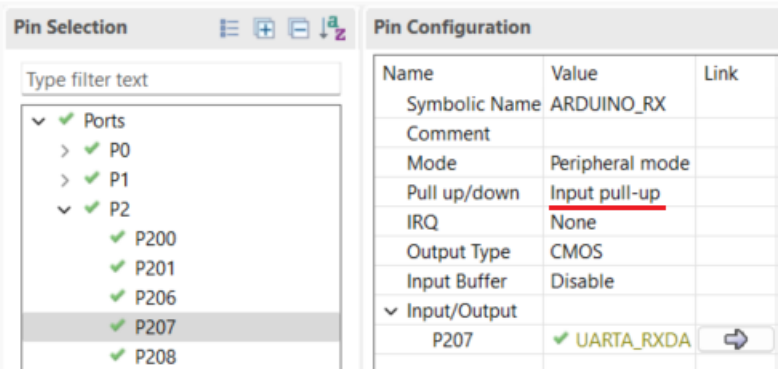
r_sau_uart 모듈의 경우 아래와 같이 설정합니다.
모율의 이름은 g_easyDSP 로 설정하고 적절히 채널을 선택합니다.
보드레이트를 설정하되, 이는 easyDSP 프로젝트에서의 보드레이트 설정값과 동일해야 합니다.
콜백함수 이름을 easyDSP_callback 으로 설정하고 인터럽트 순위는 최하위로 설정합니다 (높은 숫자).
마지막으로 핀을 할당합니다.



r_uarta 모듈의 경우 앞서와 마찬가지 개념으로 아래와 같이 설정합니다.



Pins 탭에서 사용하는 RXD, TXD 핀 모두에 풀업을 설정합니다.



사용하는 통신 채널 (SAU 또는 UARTA)에 적절한 클럭이 공급되는 지 Clocks 탭에서 확인합니다.
마지막으로 코드를 생성합니다.

easyDSP_init() 함수 호출 및 IDE 설정

다른 RA 시리즈와 동일하오니 여기 를 참조 바랍니다.

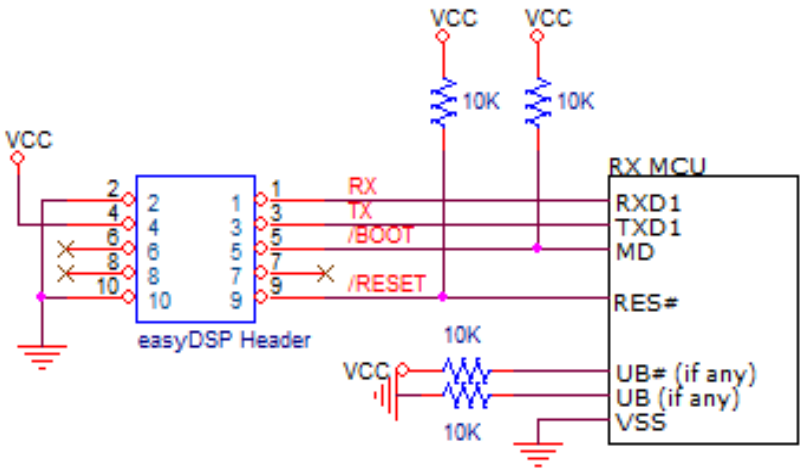
7.11 RX

7.11.1 RX 하드웨어 설정

---

easyDSP 와 연결

SCI1 을 사용해야 easyDSP 가 플래시 프로그래밍을 지원할 수 있습니다.
따라서 하기 결선과 같이 SCI1 의 RXD1, TXD1 을 easyDSP 해당핀과 연결하세요..
easyDSP 헤더 4 번 핀에는 MCU 의 VCC 를 연결하여 주십시요.



아래에서 MCU 에 따른 각 핀 참조하세요. 핀넘버는 MCU 데이타시트를 참조하시기 바랍니다.

UB, UB# 핀이 있는 MCU 의 경우 각각 풀다운/풀업 저항 연결이 필요합니다.

MCU		RXD1	TXD1	UB or UB#
RX100	RX110	P15	P16	N.A.
	RX111	P15	P16	P14/UB#
	RX113	P15	P16	P14/UB#
	RX130	P30	P26	N.A.
	RX13T	PB7	PB6	N.A.
	RX140	P30	P26	N.A.
RX200	RX230 RX231	P30	P26	PC7/UB
	RX23E-A	P30	P26	N.A.
	RX23T	PD5	PD3	N.A.
	RX23W	P30	P26	PC7/UB
	RX24T	PD5	PD3	N.A.
	RX24U	PD5	PD3	N.A.
	RX26T	PD5	PD3	N.A.
RX600	RX64M	PF0 (177/176-pin products) P26 (145/144/100-pin products)	PF2 (177/176-pin products) P30 (145/144/100-pin products)	PC7/UB
	RX651 RX65N	PF0 (177- and 176-pin products) P26 (145-, 144-, 100-, and 64-pin products)	PF2 (177- and 176-pin products) P30 (145-, 144-, 100-, and 64-pin products)	PC7/UB
	RX660	P26	P30	PC7/UB
	RX66N	PF0 (224- and 176-pin products) P26 (145-, 144-, and 100-pin products)	PF2 (224- and 176-pin products) P30 (145-, 144-, and 100-pin products)	PC7/UB
	RX66T	PD3	PD5	P00/UB
	RX671	P26	P30	PC7/UB
RX700	RX71M	PF0 (177/176-pin products) P26 (145/144/100-pin products)	PF2 (177/176-pin products) P30 (145/144/100-pin products)	PC7/UB
	RX72M RX72N	PF0 (224- and 176-pin products) P26 (144-, and 100-pin products)	PF2 (224- and 176-pin products) P30 (144-, and 100-pin products)	PC7/UB
	RX72T	PD3	PD5	P00/UB

note) N.A. = not available

기타 주의 사항 :

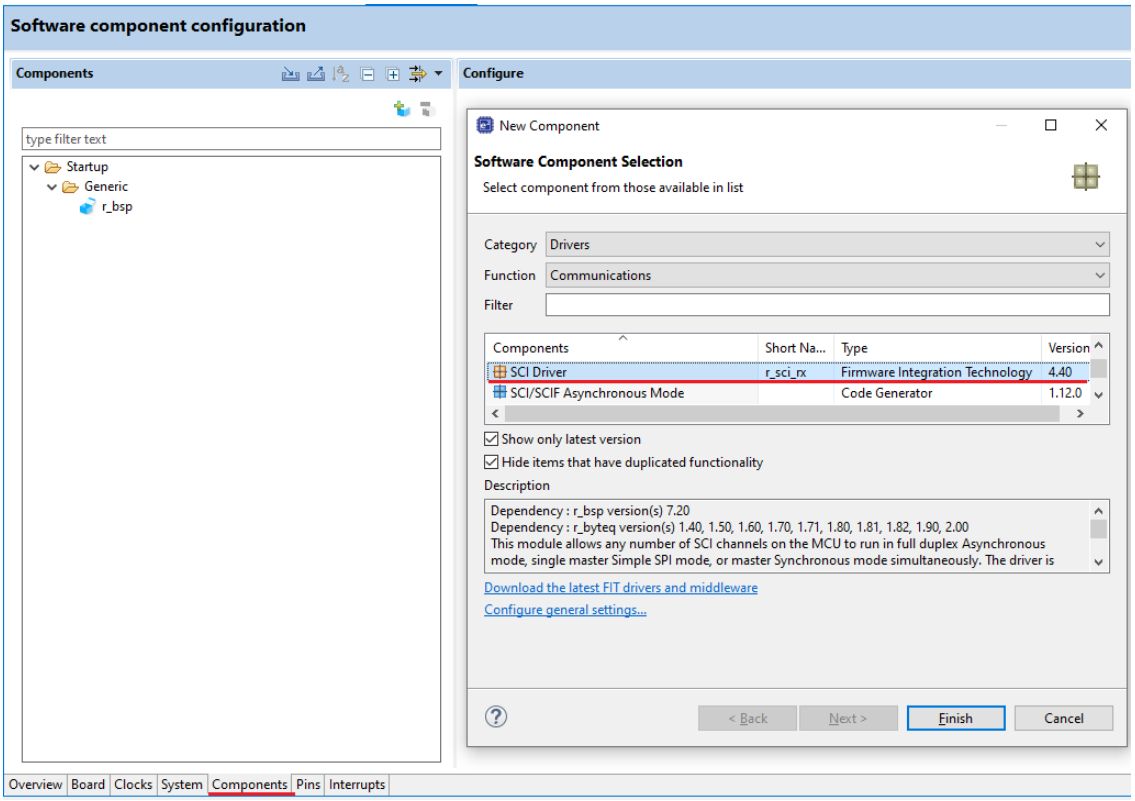
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU RES# 신호 사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 RES#에 전달해야 함
- easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업되어 있습니다.
- 만약 SCI1 을 사용할 수 없는 경우라면, 다른 SCI 채널을 사용할 수 있습니다. 하지만 이 경우 플래시 프로그래밍이 지원되지 않고, 모니터링만 되며, 또한 /BOOT 핀과 /RESET 핀 연결은 필요하지 않습니다.

7.11.2 RX 소프트웨어 설정

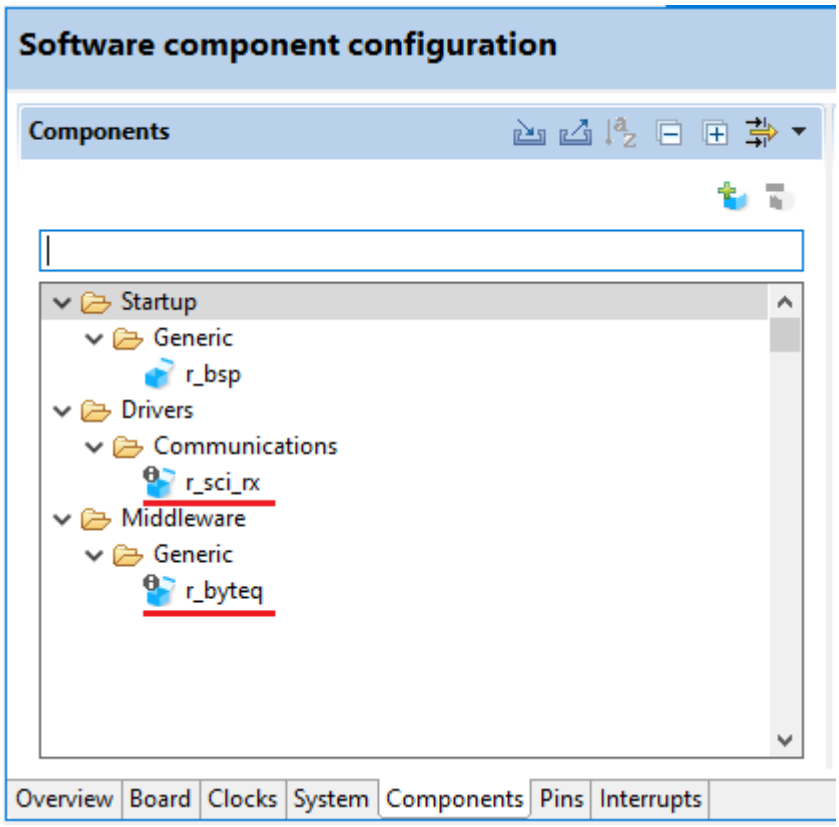
easyDSP 는 RX Smart Configurator 에서 생성된 소스 코드를 사용하고 있습니다.
하기에서는 RX Smart Configurator v1.40 기준으로 설명드립니다.

STEP 1 : Smart Configurator 설정

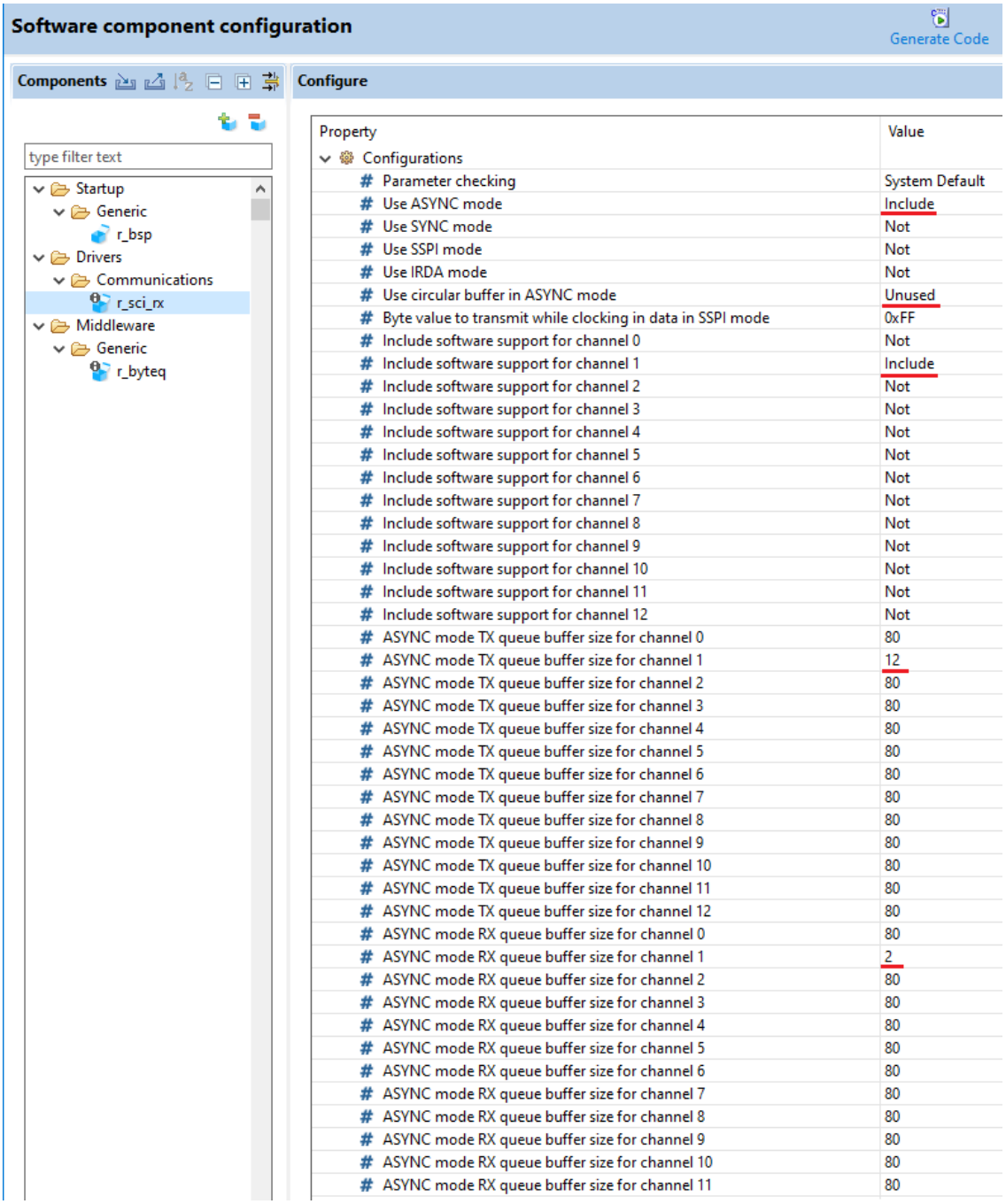
Components 탭에서 'Add component' 버튼을 누르고, SCI Driver 컴포넌트를 추가하면



아래처럼 r_sci_rx 및 r_byteq 컴포넌트가 생성됩니다.

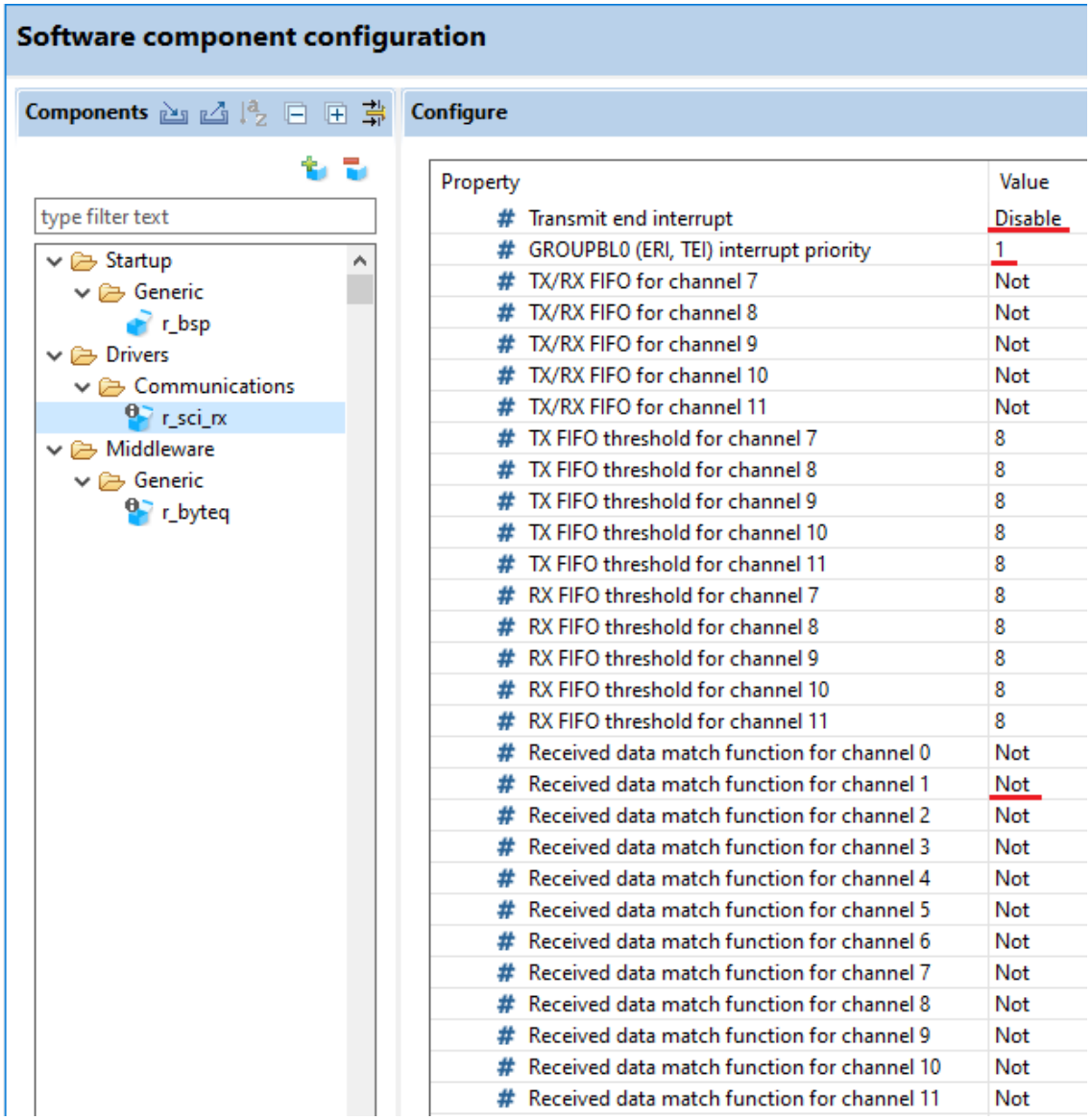


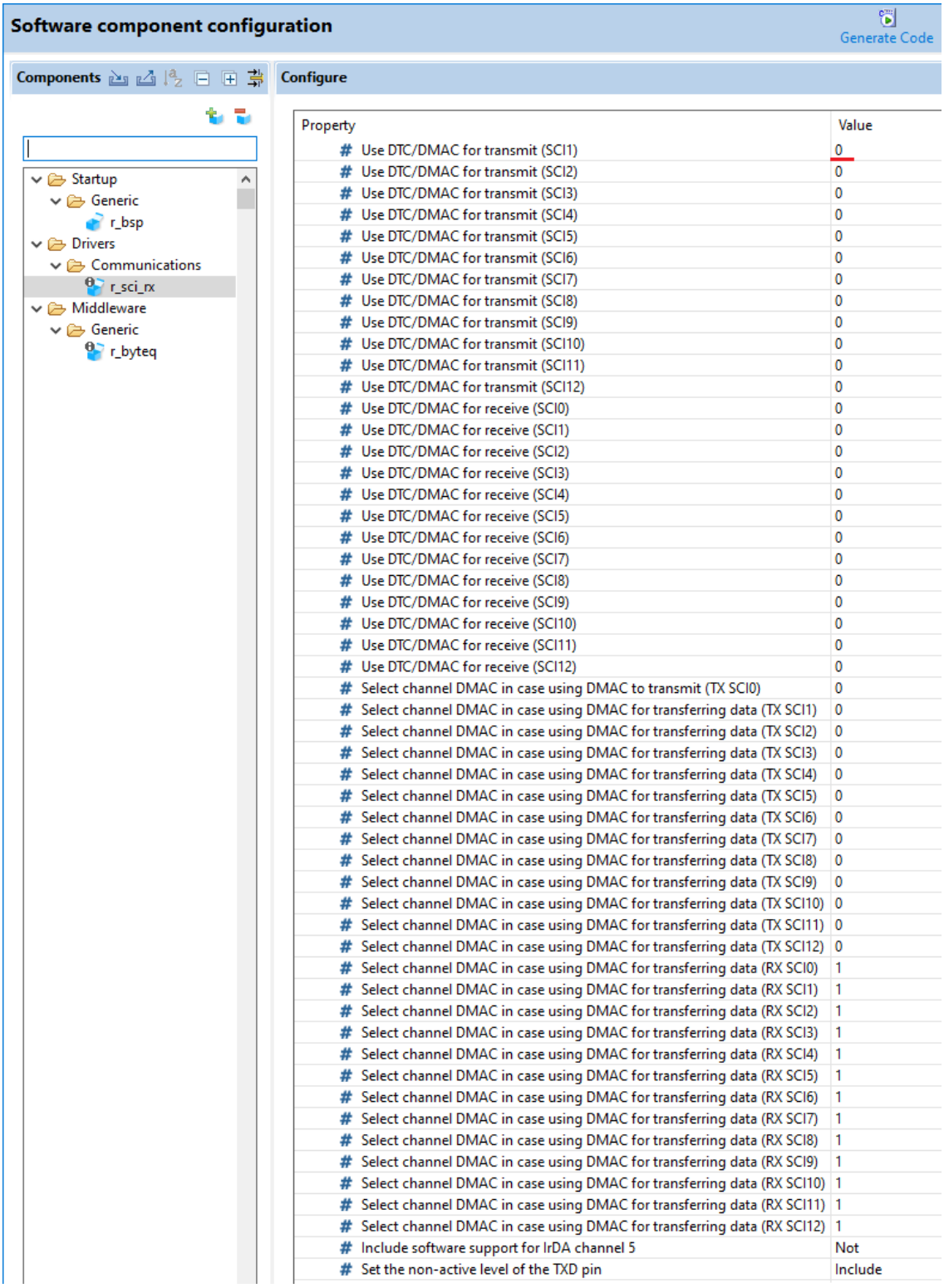
easyDSP 는 SCI 채널 1 을 사용하므로, r_sci_rx 컴포넌트에서 관련 설정이 필요합니다. 아래 그림 빨간색 부분을 참조하세요. easyDSP 용도로는 circular 버퍼를 사용할 필요가 없습니다만 만약 다른 용도로 사용하게 된다면, 추후 r_byteq 설정에서 적절한 주의가 필요합니다. 큐버퍼의 크기는 TX, RX 각각 최소 12 개, 2 개가 필요합니다.



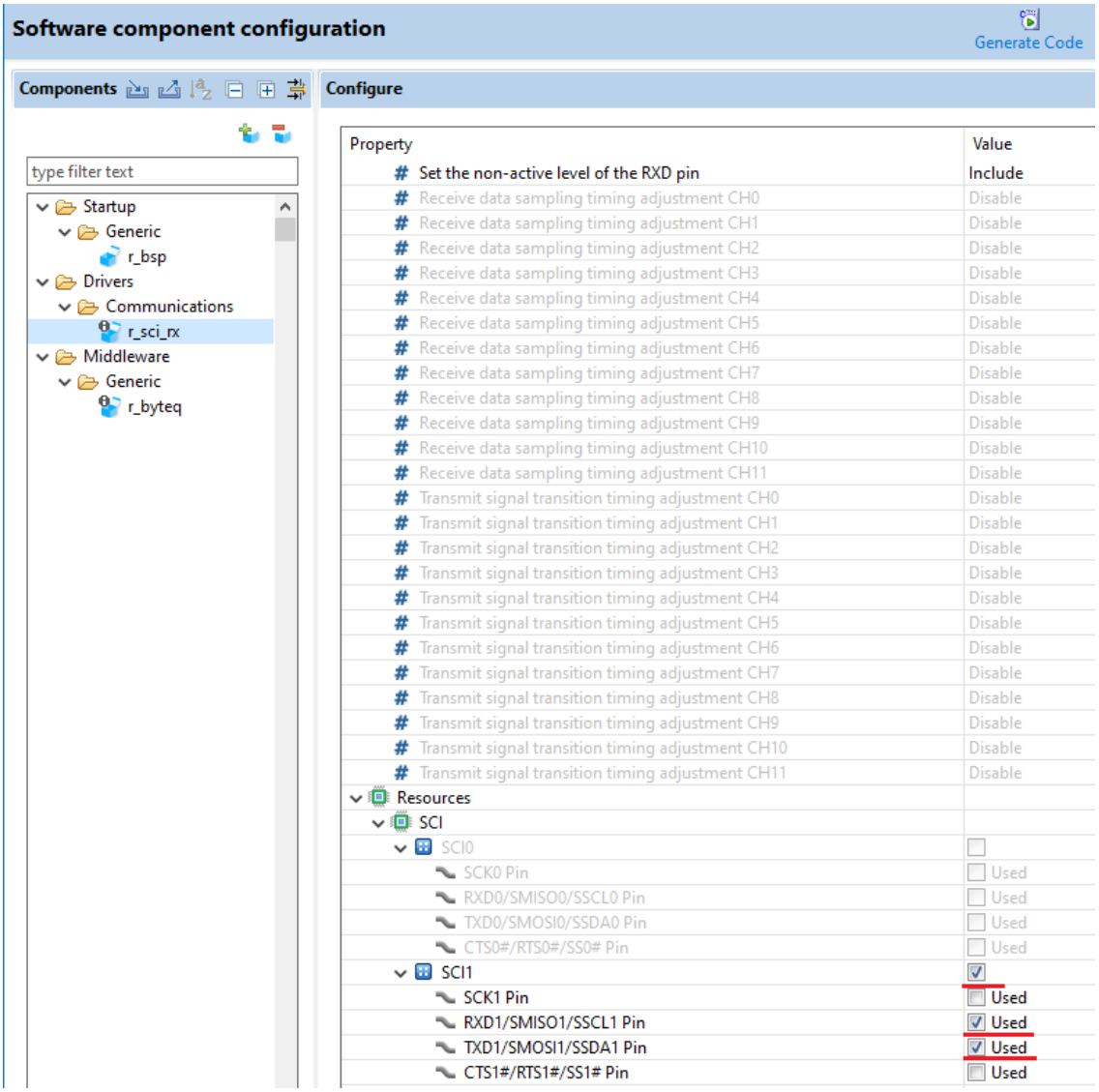
TEI 인터럽트는 사용하지 않습니다.

ERI, TEI 인터럽트 순위는 가장 낮은 1 로 설정하세요.

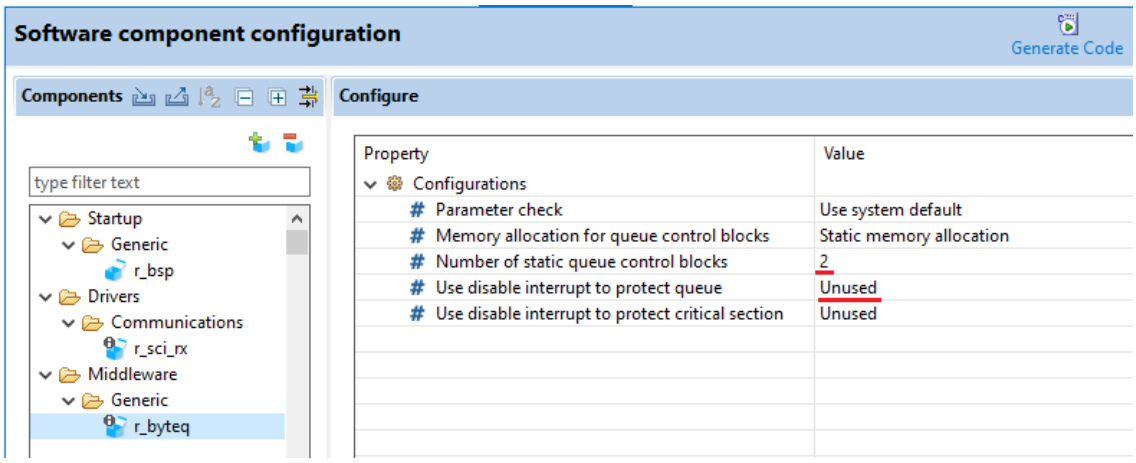

Software component configuration
Components
Configure
type filter text
Startup
Generic
r_bsp
Drivers
Communications
r_sci_rx
Middleware
Generic
r_byteq
Property
Value
Transmit end interrupt
Disable
GROUPBL0 (ERI, TEI) interrupt priority
1
TX/RX FIFO for channel 7
Not
TX/RX FIFO for channel 8
Not
TX/RX FIFO for channel 9
Not
TX/RX FIFO for channel 10
Not
TX/RX FIFO for channel 11
Not
TX FIFO threshold for channel 7
8
TX FIFO threshold for channel 8
8
TX FIFO threshold for channel 9
8
TX FIFO threshold for channel 10
8
TX FIFO threshold for channel 11
8
RX FIFO threshold for channel 7
8
RX FIFO threshold for channel 8
8
RX FIFO threshold for channel 9
8
RX FIFO threshold for channel 10
8
RX FIFO threshold for channel 11
8
Received data match function for channel 0
Not
Received data match function for channel 1
Not
Received data match function for channel 2
Not
Received data match function for channel 3
Not
Received data match function for channel 4
Not
Received data match function for channel 5
Not
Received data match function for channel 6
Not
Received data match function for channel 7
Not
Received data match function for channel 8
Not
Received data match function for channel 9
Not
Received data match function for channel 10
Not
Received data match function for channel 11
Not



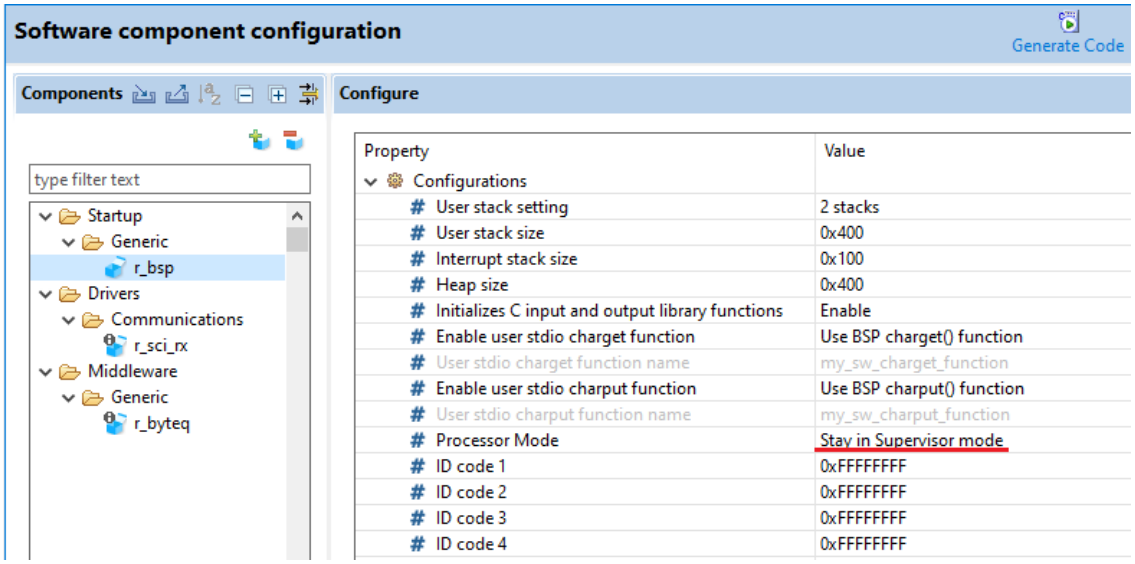
SCI1 의 RXD1, TXD1 핀을 활성화시킵니다. 다른 SCI1 핀은 사용하지 않습니다.



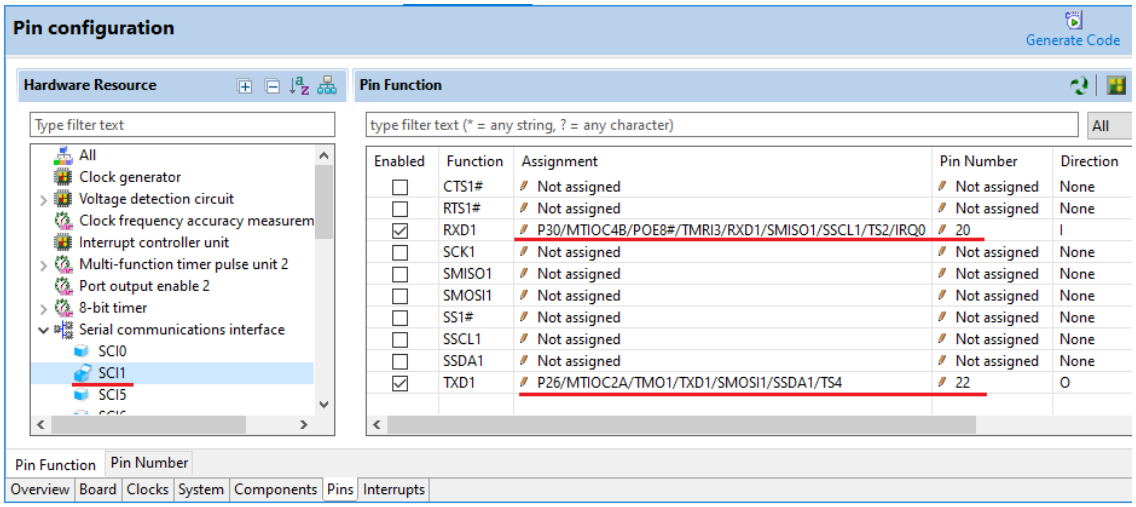
다음은 r_byteq 컴포넌트 설정입니다. easyDSP 통신을 위해 최소 2 개의 queue control blocks 이 필요합니다. Circular 버퍼를 사용하지 않는다면 'Use disable interrupt to protect queue' 값을 Unused 로 설정합니다. Circular buffer 사용시 'Used'로 설정합니다.



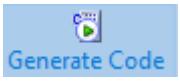
r_bsp 컴포넌트의 Process Mode 를 Stay in Supervisor mode 로 설정해야 합니다.



Pins 탭에서 RXD1, TXD1 핀을 할당합니다. 'Assignment'에서의 설정은 RX 하드웨어 설정 과 매칭시켜 주세요. Pin Number 설정은 MCU 데이타시트를 참조하여 선택하시기 바랍니다.



마지막으로 코드를 생성합니다.



STEP 2 : easyDSP_init() 함수 호출

먼저 easyDSP 통신을 위해 제공되는 소스파일 (easyRX.h, easyRX.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩RX 에서 찾을 수 있습니다.

먼저 easyRX.h 파일에서 easyDSP 통신에 사용할 SCI 채널의 보드레이트를 설정하시기 바랍니다. 이 값은 easyDSP 프로젝트 설정에서 지정할 보드레이트와 서로 일치해야 합니다.

```
//////////////////////////////////////////////////////////////
// set the baud rate for UART communication with easyDSP
// it should be same to the baudrate of easyDSP project
//////////////////////////////////////////////////////////////
#define EZ_BUAD_RATE        230400
```

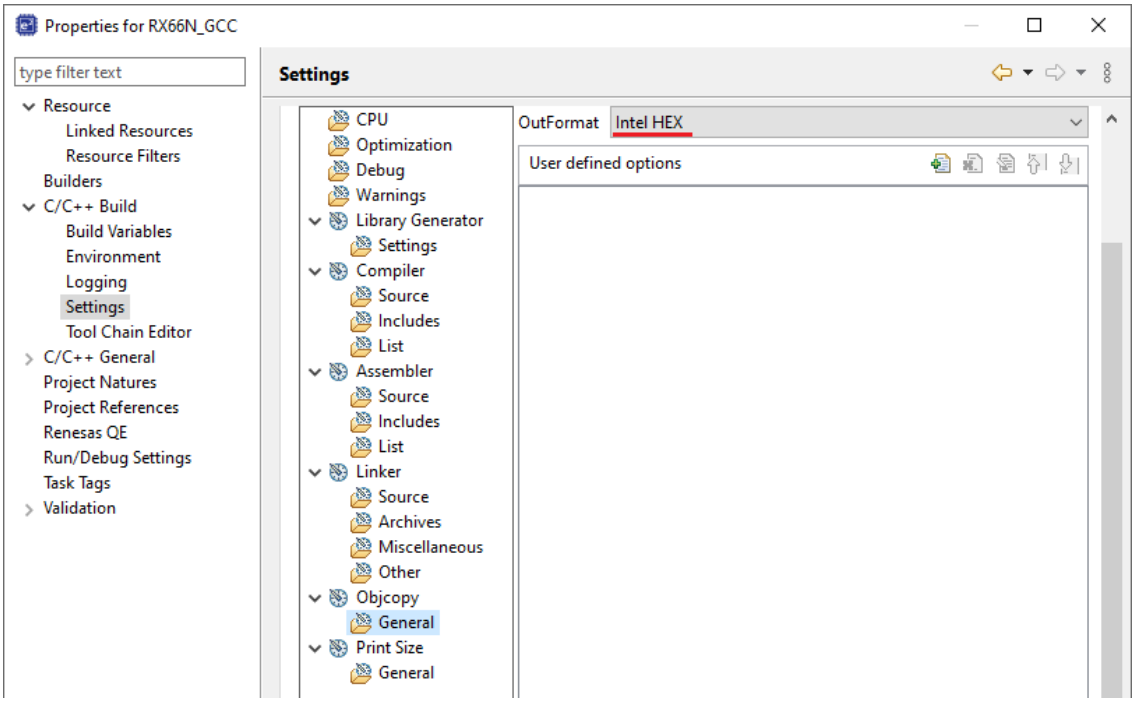
그리고 하기와 같이 main.c 에서 easyDSP_init()를 호출하여 주십시요.
easyDSP 가 사용하는 SCI 채널 관련 인터럽트의 순위는 가장 낮은 1 로 설정됩니다. 따라서 다른 사용자 인터럽트의 순위는 이보다 높은 값으로 설정하세요.

```
#include "easyRX.h"

int main(void)
{
    // initial setting
    .
    .
    .
    .
    .
    .

    // call easyDSP_init() to enable easyDSP monitoring
    easyDSP_init();

    // loop forever
    while(1)
    {
        .
        .
        .
    }
}
```

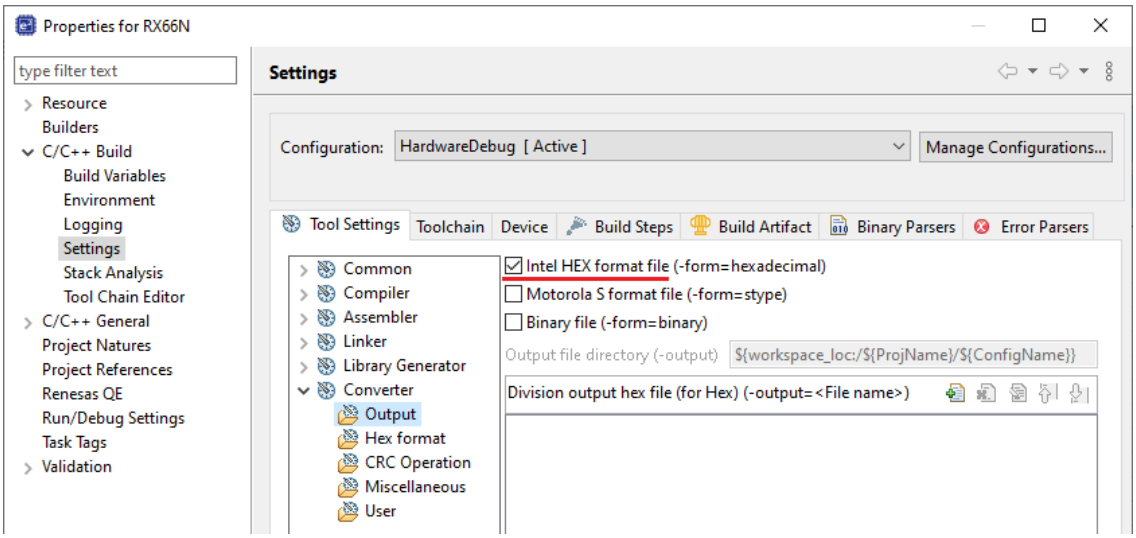
STEP 3 : IDE 설정

1. easyDSP 에서 사용하는 출력 파일(easyDSP 프로젝트에서 선정)은 DWARF 형식 디버깅 정보를 가지는 파일이어야 합니다. 따라서 CC-RX 컴파일러를 사용하실 경우, 매 컴파일마다 DWARF 디버깅 정보를 가지는 출력 파일을 생성시켜 줘야 합니다. e2 studio 경우 기본 IDE 설정에서 이를 지원하며, *.x 확장자로 출력 파일이 생성됩니다.
2. 매 컴파일마다 hex 파일(인텔 형식)이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 개발 환경을 설정 해주세요. hex 파일은 플래시 프로그래밍할 때 사용됩니다.
   Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.

예를 들어 e2 stuio 에서 GCC 컴파일러 사용 경우 :



또는 e2 studio 에서 CC-RX 컴파일러 사용 경우 :

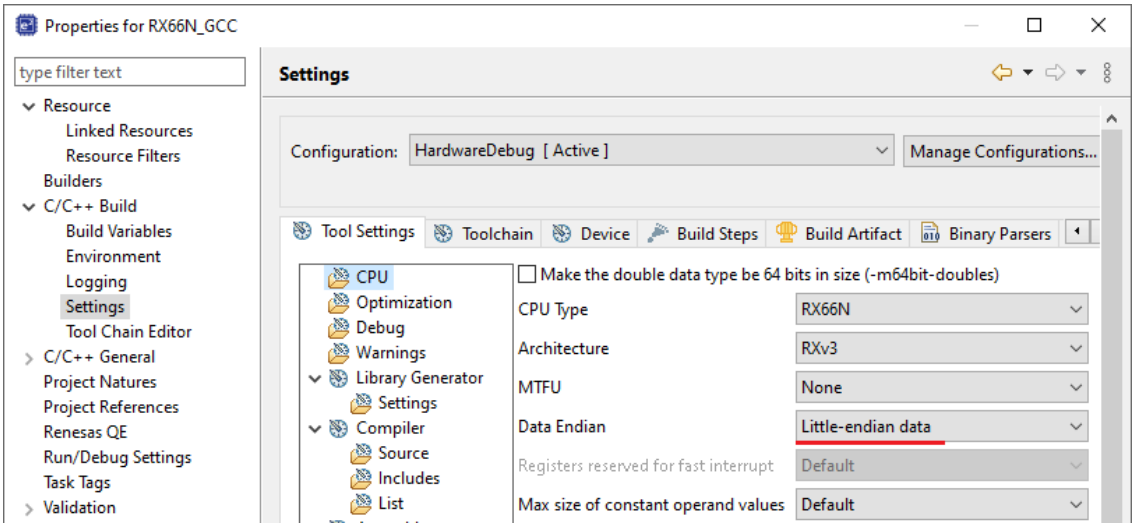


3. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.

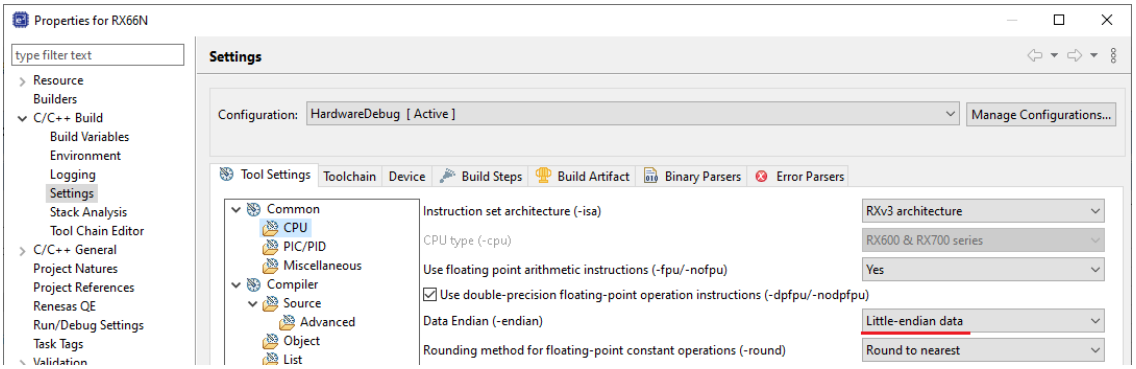
4. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수가 포함되게 하기 위해서라면 적절히 옵션을 변경하시기 바랍니다.

5. easyDSP 는 little endian 형식만 지원합니다.

e2 stuio 에서 GCC 컴파일러 사용 경우 :



또는 e2 studio 에서 CC-RX 컴파일러 사용 경우 :

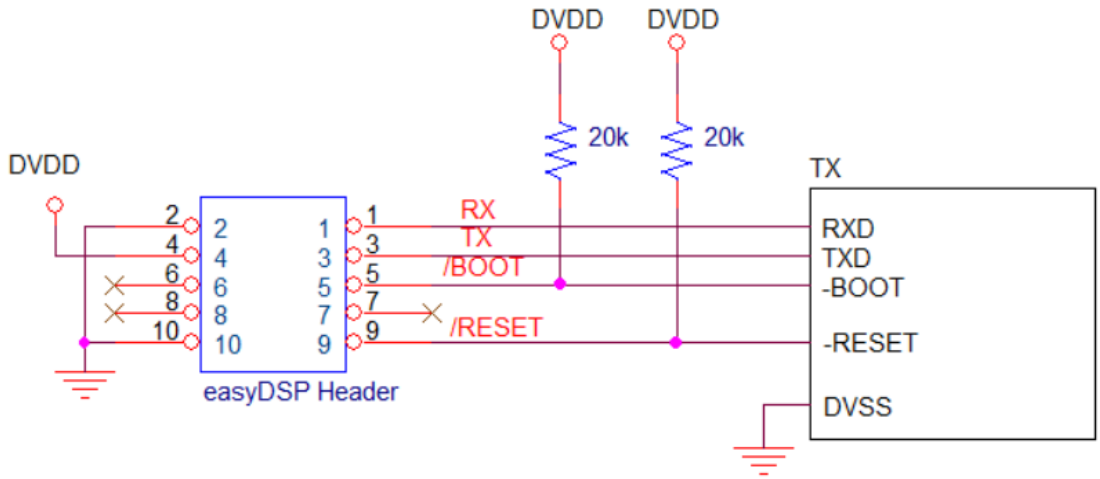


7.12 TX

TX 설정

STEP 1 : 하드웨어 설정

easyDSP 는 플래시 라이팅을 위해 Single Boot Mode 를 사용합니다. 따라서 Single Boot Mode 에서 지원되는 SIO/UART 채널을 사용해야 합니다.
다른 채널을 사용할 경우 easyDSP 는 모니터링은 지원하나, 플래시 라이팅은 지원하지 않음에 유의 부탁드립니다.
데이타시트를 확인하셔서, Single Boot Mode 에서 지원되는 SIO/UART 채널의 TXD/RXD 및 부트핀을 하기 그림과 같이 연결하시기 바랍니다.



예를 들어 TMPM370FY 의 경우 데이타시트의 해당 부분 캡쳐는 아래와 같으며, easyDSP 포드의 /BOOT 핀은

PF0 핀에, TX 핀은 PE0 핀에, RX 핀은 PE1 핀에 연결하시기 바랍니다.

19.2.6 Interface specification

In Single Boot mode, an SIO channel is used for communications with a programming controller. The same configuration is applied to a communication format on a programming controller to execute the on-board programming. Both UART (asynchronous) and I/O Interface (synchronous) modes are supported. The communication formats are shown below.

- UART communication

  Communication channel : SIO channel 0

  Serial transfer mode : UART (asynchronous) mode, half -duplex, LSB first

  Data length : 8 bits

  Parity bit : None

  STOP bit : 1 bit

  Baud rate : Arbitrary baud rate

- I/O Interface mode

  Communication channel : SIO channel 0

  Serial transfer mode : I/O interface mode, full -duplex, LSB first

  Synchronization clock (SCLK0) : Input mode

  Handshaking signal : PE4 configured as an output mode

  Baud rate : Arbitrary baud rate

Table 19-3 Required Pin Connections

Pin		Interface	
		UART	I/O Interface Mode
Power supply pins	DVDD5	o	o
	DVDD5E	o	o
	DVSS	o	o
	AVDD5A	o	o
	AVSSA	o	o
	AVDD5B	o	o
	AVSSB	o	o
	VOUT3	o	o
	VOUT15	o	o
	RVDD5	o	o
Mode-setting pin	$\overline{\text{BOOT}}$ (PF0)	o	o
Reset pin	$\overline{\text{RESET}}$	o	o
Communication pin	TXD0 (PE0)	o	o
	RXD0 (PE1)	o	o
	SCLK0 (PE2)	×	o (Input mode)
	PE4	×	o (Output mode)

기타 주의 사항 :

- DVDD = DVDD3 또는 DVDD5 (MCU 에 따라)
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU -RESET 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 -RESET 에 전달해야 함

- easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업되어 있습니다.
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함

STEP 2 : easyDSP 제공 헤더 파일 수정

먼저 easyDSP 통신을 위해 제공되는 소스파일 (easyTX.h, easyTX.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩TX_TXZ 에서 찾을 수 있습니다.
본 파일에서 업체 제공 Peripheral Driver 라이브러리를 사용하고 있으므로 해당 라이브러리도 프로젝트에 포함되어 있어야 합니다.

우선, 사용하는 MCU 에 맞춰 2 가지 파일(*_gpio.h 와 *_uart.h)을 include 하여 줍니다.
또한, 상기 언급된 하드웨어 결선에 맞춰, 채널 번호 및 통신 핀을 설정합니다.
하기 예제는 TMPM370 을 기반으로 작성되었으므로 다른 MCU 에 대해서는 적절히 변경 필요합니다.
마지막으로 easyDSP 통신 보드레이트를 설정하시기 바랍니다. easyDSP 프로젝트에서 사용될 보드레이트와 동일하게 설정합니다.

```
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
// step 1 : change header files(*_gpio.h and *_uart.h) according to MCU
//          and set the SIO/UART channel number and its port pins.
////////////////////////////////////////////////////////////////////////////////

// for TMPM370 : SIO/UART channel 0 /w PE0 and PE1 port
#include "tmpm370_gpio.h"
#include "tmpm370_uart.h"
#define EZ_UARTx_CH         0
#define EZ_SIO_PORT         GPIO_PE
#define EZ_TXD_BIT_NUM      GPIO_BIT_0
#define EZ_RXD_BIT_NUM      GPIO_BIT_1


////////////////////////////////////////////////////////////////////////////////
// step 2 : set the baud rate for SIO/UART communication with easyDSP
//          it should be same to the baudrate of easyDSP project
////////////////////////////////////////////////////////////////////////////////
#define EZ_BUAD_RATE        230400
```

STEP 3 : easyDSP_init() 함수 호출

먼저 main.c 상단에 easyTX.h 를 include 하여 주시고,
main 함수 적절 부분에 easyDSP_init() 함수를 호출하시기 바랍니다.

```
#include "easyTx.h"

int main(void)
{




    easyDSP_init();

    while (1) {

    }
}
```

STEP 4 : IDE 설정

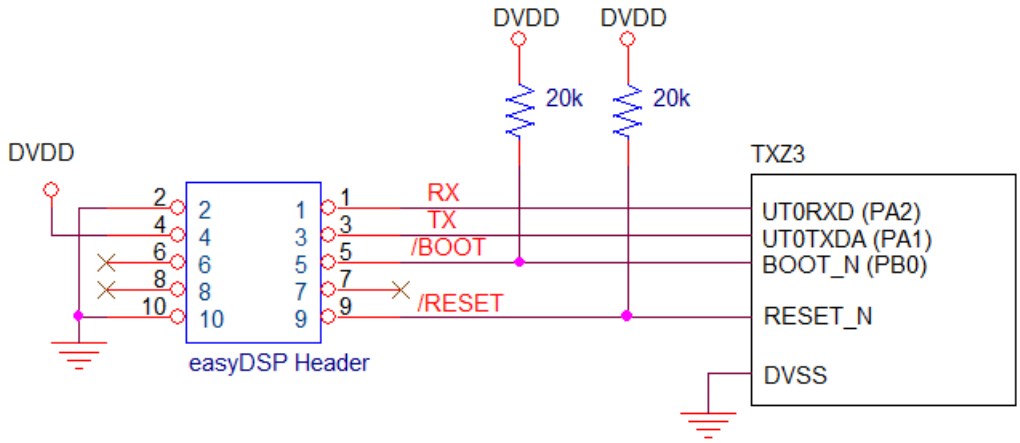
1. 매 컴파일마다 hex 파일 (인텔 형식) 이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 IDE 를 설정해주세요. Hex 파일은 플래시 프로그래밍할 때 사용됩니다.
   Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.
2. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.
3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수를 포함되게 하기 위해서라면 해당하는 세팅이 필요합니다.
4. easyTx.c 에서 인라인 함수를 사용하므로, 필요시 컴파일러 옵션 c99 모드를 활성화시켜 주세요.

7.13 TXZ3

TXZ3 설정

STEP 1 : 하드웨어 설정

easyDSP 는 플래시 라이팅을 위해 Single Boot Mode 를 사용합니다. 따라서 Single Boot Mode 에서 지원되는 UART0 채널, 그 중에서도 PA1/PA2 을 사용해야 합니다.
다른 채널을 사용할 경우 easyDSP 는 모니터링은 지원하나, 플래시 라이팅은 지원하지 않으며, 또한 easyDSP 소스파일 easyTXZ3.c 를 해당 채널에 맞게 직접 수정하셔야 합니다.
하기 그림과 같이 연결하시기 바랍니다.



기타 주의 사항 :

- DVDD = DVDD3 또는 DVDD5
- /RESET 핀은 MCU 에 리셋을 줄 수 있도록 적절히 연결 (/RESET 핀의 Low 상태 유지 기간은 약 500msec)
- easyDSP /RESET 신호와 MCU -RESET 신호사이에 리셋 IC 같은 회로가 삽입된다면, 삽입된 회로는 /RESET 신호를 0.5 초내에 -RESET 에 전달해야 함
- 각 신호선에 풀업 저항을 설치할 시에 그 값이 수 KOhm 이상이 되어야 함
- easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업되어 있습니다.

STEP 2 : easyDSP 제공 헤더 파일 수정

먼저 easyDSP 통신을 위해 제공되는 소스파일 (easyTXZ3.h, easyTXZ3.c)을 프로젝트에 포함하시기 바랍니다.
해당 파일은 easyDSP 프로그램이 인스톨된 폴더에서 ₩source₩TX_TXZ 에서 찾을 수 있습니다.

사용 환경에 맞춰 ezsyTXZ3.h 초기 부분을 설정하시기 바랍니다. 우선, 사용하는 MCU 에 맞춰 CMSIS 헤더파일을 인클루드하여 줍니다.
마지막으로 easyDSP 통신 보드레이트를 설정하시기 바랍니다. easyDSP 프로젝트에서 사용될 보드레이트와 동일하게 설정합니다.

```
////////////////////////////////////////////////////////////////////////////////
// step 1 : include CMSIS Peripheral Access Layer Header File
//          include header file name accordingly to target MCU
////////////////////////////////////////////////////////////////////////////////
#include <TMPM3H6.h>    // For example, TMPM3H6.h for TMPM3H6 MCU.
//#include <TMPM3HQ.h>  // For example, TMPM3HQ.h for TMPM3HQ MCU.

////////////////////////////////////////////////////////////////////////////////
// step 2 : set the baud rate for UART communication with easyDSP
//          it should be same to the baudrate of easyDSP project
////////////////////////////////////////////////////////////////////////////////
#define EZ_BUAD_RATE        115200
```

STEP 3 : easyDSP_init() 함수 호출

먼저 main.c 상단에 easyTXZ3.h 를 include 하여 주시고,
main 함수 적절 부분 (기타 초기화 루틴 이후)에 easyDSP_init() 함수를 호출하시기 바랍니다.

```
#include "easyTXZ3.h"

int main(void)
{
    .
    .
    .
    .
    .

    easyDSP_init();

    while (1){


    }
}
```

STEP 4 : IDE 설정

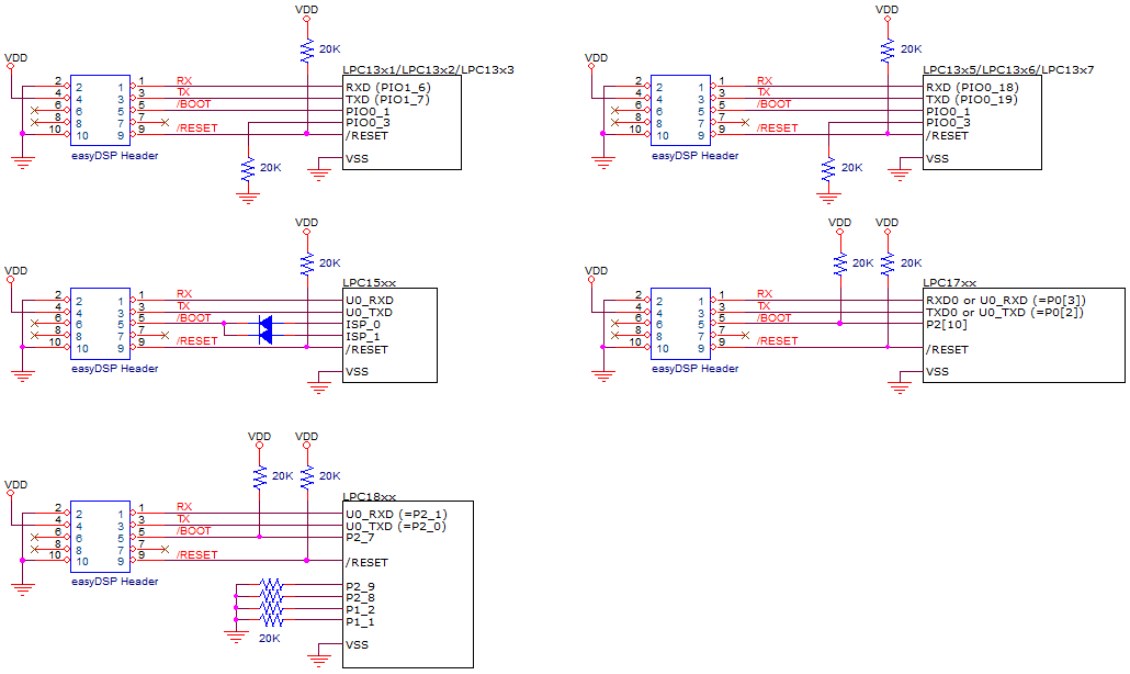
1. 매 컴파일마다 hex 파일 (인텔 형식) 이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 IDE 를 설정해주세요. Hex 파일은 플래시 프로그래밍할 때 사용됩니다.
   Hex 파일 확장자는 hex 또는 ihex 가 될 수 있습니다. easyDSP 는 확장자 hex 파일의 존재를 먼저 확인하여 사용하고, 존재하지 않을 경우 확장자 ihex 파일을 사용합니다.
2. easyDSP 로 변수를 억세스하기 위해서는, 출력 파일(예:*.elf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.
3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수를 포함되게 하기 위해서라면 해당하는 IDE 세팅이 필요합니다.
4. easyTXZ3.c 에서 인라인 함수를 사용하므로, 필요시 컴파일러 옵션 c99 모드를 활성화시켜 주세요.

7.14 LPC

LPC1x00 설정

STEP 1 : 하드웨어 설정

easyDSP 는 MCU 플래시 프로그래밍을 지원하기 위해 ISP 가 지원되는 USART (또는 UART) 채널 0 번을 사용합니다. MCU 종류에 따라 하기 그림과 같이 연결하시기 바랍니다.



LPC1500 의 경우 MCU 패키지에 따른 핀에 대해서 하기 테이블 참조하세요.

ISP pin	LQFP48	LQFP64	LQFP100
ISP_0	PIO0_4	PIO1_9	PIO2_5
ISP_1	PIO0_16	PIO1_11	PIO2_4
U0_TXD	PIO0_15	PIO0_18	PIO2_6
U0_RXD	PIO0_14	PIO0_13	PIO2_7

LPC1800 의 경우, 부트 모드가 부트 핀에 의해 결정되도록 MCU 가 설정되어야 합니다. 즉, OTP 메모리가 프로그램되어 있지 않거나, OTP BOOT_SRC bits 이 모두 0 이어야 합니다 (출하 기준 기본값).

기타 주의 사항 : easyDSP 헤더 RX, TX 신호는 easyDSP 포드 내부에서 100k 오옴으로 풀업

STEP 2 : LPCOpen 라이브러리 포함

easyDSP 는 NXP 가 제공하는 LPCOpen 라이브러리를 사용하여 USART 통신 기능을 구현하고 있습니다. 따라서 사용자 프로그램에서도 해당 라이브러리가 포함되도록 설정하세요..

STEP 3 : easyDSP 제공 소스파일 포함

easyDSP 에서 제공하는 소스파일 (easyLPC1x00_va.b.h, easyLPC1x00_va.b.c)을 프로젝트에 포함하시기 바랍니다.
(a, b = 버전에 따라 달라질 수 있음)
해당 파일은 easyDSP 프로그램이 설치된 폴더안에 source₩LPC 폴더에서 찾을 수 있습니다.

MCU 종류에 따라 별도의 소스 파일이 사용됩니다 (예를 들어 LPC1700 경우 easyLPC1700.c)

사용 환경에 맞춰 헤더파일 ezsyLPC1x00.h 초기 부분을 설정하시기 바랍니다.
MCU 종류에 따라 설정 내용이 달라지며, 타켓 MCU 를 설정하거나, 타켓 MCU 의 패키지를 설정하거나,
easyDSP 와의 통신 보드레이트을 설정합니다.
보드레이트는 easyDSP 프로젝트에서 사용될 보드레이트와 동일하게 설정합니다.
아래 LPC1500 의 경우 참조하세요.

```
////////////////////////////////////////////////////////////////////////////////////////////
// Package Selection : Please choose MCU package and define it as 1. set as 0 for others.
////////////////////////////////////////////////////////////////////////////////////////////
#define EZDSP_LQFP48            0
#define EZDSP_LQFP64            1
#define EZDSP_LQFP100           0

////////////////////////////////////////////////////////////////////////////////////////////
// step 2 : set the baud rate for USART communication with easyDSP
//          it should be same to the baudrate of easyDSP project
////////////////////////////////////////////////////////////////////////////////////////////
#define EZDSP_BAUDRATE           115200L
```

STEP 4 : easyDSP 초기화 함수 호출

먼저 main.c 상단에 easyLPC1x00_va.b.h 를 include 하여 주시고, main 함수 초기화 루틴 이후 easyDSP_init() 함수를 호출하시기 바랍니다.
easyDSP_init() 함수에서는 easyDSP 와의 통신을 위한 각종 설정을 수행합니다.

```
#include "easyLPC1x00.h"

int main(void)
{
    // initial setting
    .
    .
    .
    .
    .
    .

    // call easyDSP_init() to enable easyDSP monitoring
    easyDSP_init();

    // loop forever
    while(1)
    {
        .
        .
        .
    }
}
```

STEP 5 : IDE 설정

1. 매 컴파일마다 hex 파일(인텔 형식)이 생성되어 출력 파일과 동일한 폴더에 동일한 이름으로 위치하도록 개발 환경을 설정 해주세요. hex 파일은 플래시 프로그래밍할 때 사용됩니다.
   예를 들어 MCUXpresso IDE 를 사용할 경우, arm-none-eabi-objcopy -O ihex "${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.hex" 를 Post-build steps 에 등록해주세요.
2. easyDSP 로 모니터링을 수행하기 위해서는, 출력 파일 (예:*.axf)에 debug information 이 반드시 포함되어야 합니다.이를 위해 어셈블리/ 컴파일러/링커 옵션을 적절히 선택하시기 바랍니다.
3. 최적화 또는 링커 세팅에 따라, 선언되었지만 실제 사용되지 않는 변수는 debug information 에 포함되지 않아 easyDSP 에서 모니터링되지 않을 수 있습니다. 이 경우에도 변수를 포함되게 하기 위해서라면 해당하는 IDE 세팅이 필요합니다.

7.15 주의 사항

리셋단자(easyDSP pod #9 번핀)의 사용시 각별한 주의를 요합니다

비록 easyDSP 가 /Reset 핀을 통해 리셋을 지원합니다만, 부적절한 상황(시스템 동작 중에 easyDSP pod 를 연결하거나 연결된 pod 를 뺄 때 순간적인 리셋단자의 신호 불안, 노이즈 유입 등)에서 MCU 를 의도되지 않은 리셋 모드로 진입시킬 수 있습니다. 이러한 순간적인 리셋단자의 전위 불안을 방지하기 위해서, 사용자의 MCU 보드상에서 MCU 의 리셋단자에 충분한 필터를 장착하는 것이 필요합니다. 이러한 필터가 장착되는 상황을 고려하여, easyDSP

pod 의 리셋단자는 리셋시 약 500ms 동안 Low 상태를 유지합니다. 하지만 이러한 조치를 취해도 MCU 동작 중 easyDSP 를 물리적으로 연결하게 되면 언제든 의도치 않은 리셋이 발생할 수 있다는 점에 주의 부탁 드립니다. 어쩔 수 없이 MCU 동작 중 easyDSP 를 연결해야 한다면 easyDSP 를 먼저 PC 에 연결하고, 나중에 DSP 에 연결하시기 바랍니다. easyDSP 을 분리할 경우라면, 먼저 DSP 에서 분리, 나중에 PC 에서 분리하여 주세요.

적당한 통신 속도 (BPS)는?

사용자 보드에 따라 적절히 선택해야 합니다. 물론 통신 BPS 가 높을 수록 빠른 통신이 이루어질 것으로 기대됩니다만, 선결되어야 할 조건이 있습니다. easyDSP 가 일련의 데이타들을 특정 BPS 로 MCU 에 전송할 시에, MCU 는 이 일련의 데이타들을 원활히 받아드릴 수 있어야 합니다. 만약 115200BPS 로 일련의 데이타를 easyDSP 가 MCU 에 보낼 시에, 한 바이트가 전송되는 시간은 약 86usec(1/115200bps*10bit) 입니다. 86usec 마다 RX port 에 데이타가 전송될 때, MCU 에서 해당 인터럽트 루틴이 원활히 수행되어야 합니다. 만약 프로그램의 다른 루틴을 수행하느라 86usec 마다 RX 인터럽트 루틴이 원활히 수행되지 못한다면, 결국 통신이 실패하게 됩니다. 이러한 통신 실패가 매우 심할 경우, 결국 모든 통신이 실패되고 easyDSP 상의 모든 데이타가 '?'로 표기됩니다.

다양한 개발 환경에 대한 주의

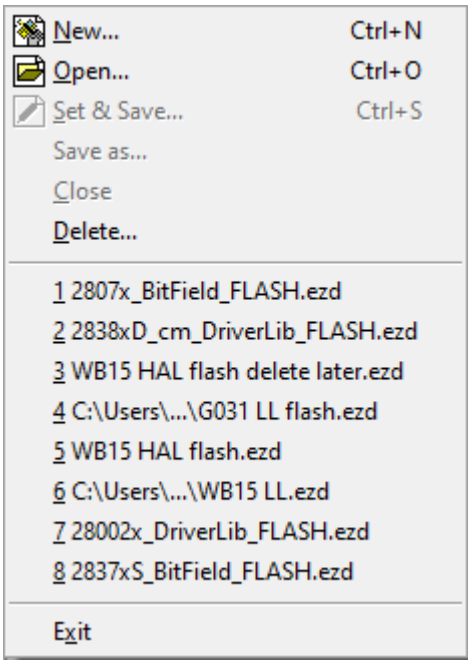
easyDSP 는 ARM 계열 MCU 의 다양한 개발 환경 을 지원하도록 설계되었지만, 모든 개발 환경에 대해서 테스트되지는 않았습니다. 만약 특정 개발 환경에서 easyDSP 가 제대로 동작하지 않는 다면 easyDSP@gmail.com 으로 버그 리포팅 부탁 드립니다.

변수가 보이지 않을 경우

컴파일러/링커 옵션 및 최적화에 따라 선언되었지만 의미있게 사용되지 않는 변수가 삭제될 수 있습니다. 이 경우 컴파일러 출력 맵 파일에서도 해당 변수가 아예 보이지 않거나 변수 주소가 0 번지로 지정되며 easyDSP 에서 변수 등록이 되지 않습니다.
해당 변수를 활성화하기 위해서는 컴파일러/링커 옵션을 변경 바랍니다.

8. 메뉴

8.1 Project



easyDSP 는 프로젝트로 파일을 관리합니다. 프로젝트 메뉴의 부속 메뉴의 해당 기능은 다음과 같습니다.

New 메뉴 : 새로운 프로젝트 생성

Open 메뉴 : 기존 프로젝트 열기

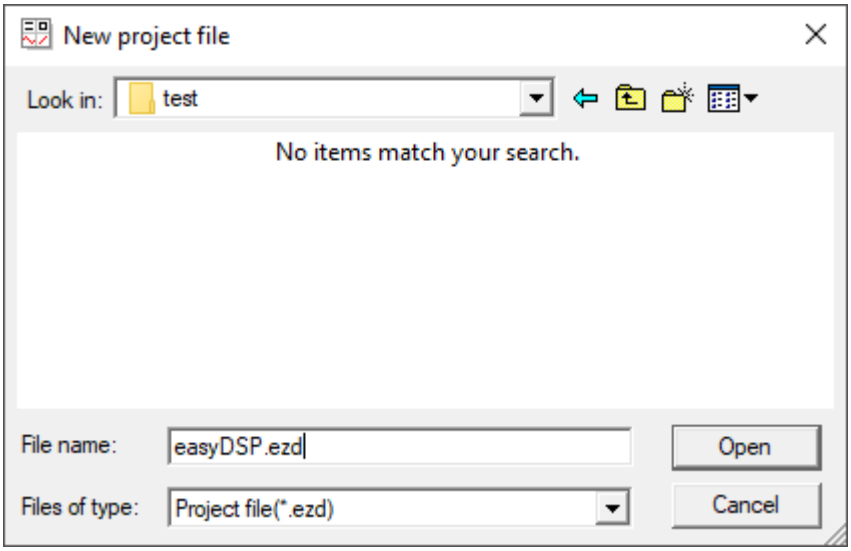
Set & Save 메뉴: 현재 프로젝트를 다른 폴더에 저장한 후 열기

Close 메뉴 : 현재 프로젝트를 닫는다

Delete 메뉴 : 프로젝트에 관련된 파일을 지운다

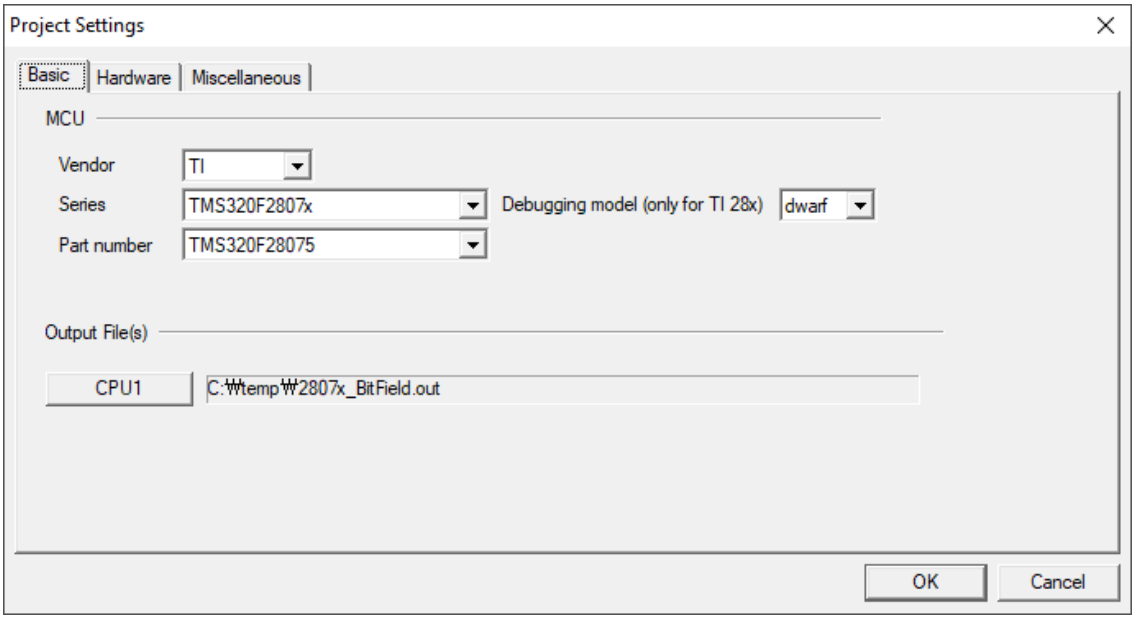
New 메뉴:

New 메뉴 실행 후 다음과 같이 프로젝트 파일 이름을 설정하는 대화상자가 나타납니다. 선정한 폴더에서 프로젝트 파일 이름을 설정 하십시요. 프로젝트 파일의 기본 확장자는 "ezd"입니다.

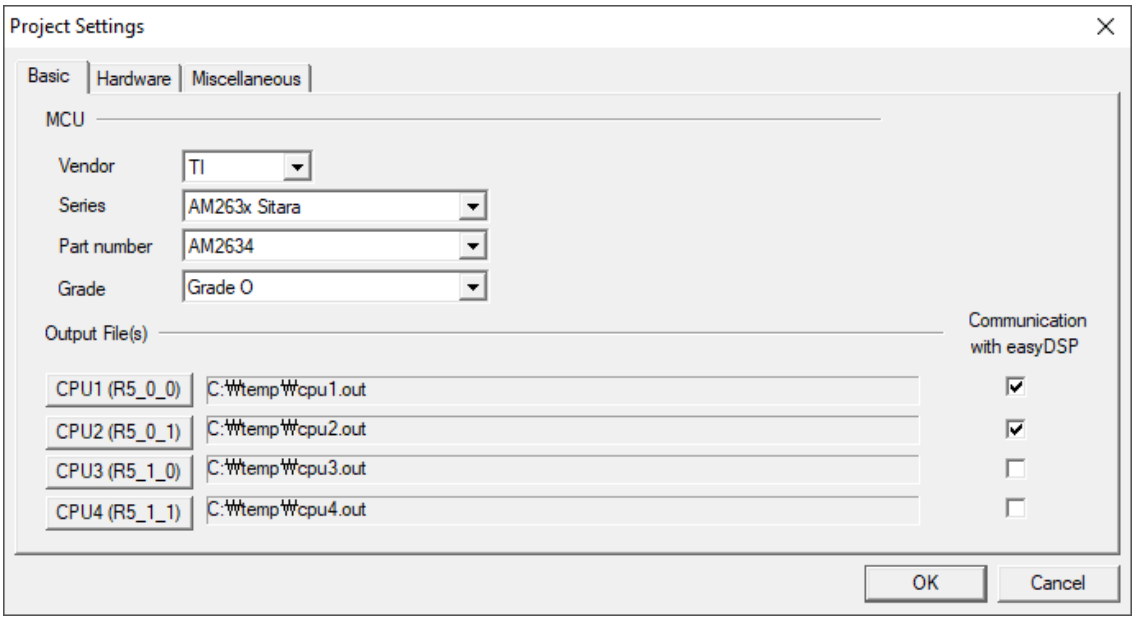


프로젝트 파일 이름을 선정한 후, 다음과 같은 속성시트를 통하여 프로젝트를 설정합니다.속성 시트는 'Basic', 'Hardware', 'Miscellaneous'의 속성 페이지로 구성되어 있습니다.

'Basic' 속성 페이지에서 프로젝트의 기본 정보를 설정합니다.
먼저 사용할 MCU 를 선정합니다.
STM32 의 경우 MCU 이름에 single/dual bank 설정이 포함된 경우가 있는데, 이는 single 또는 dual bank 를 구분해야 하는 경우에만 포함됩니다. 즉, 항상 single bank 이거나 항상 dual bank 이거나 또는 single 및 dual bank 를 구분할 필요가 없는 경우에는 표시되지 않습니다.
TI 사 C28x 일부 MCU 의 경우 'Debugging model'을 선정해야 하며 이 경우 해당 박스가 표시됩니다. 컴파일러의 사용 옵션과 동일하게 맞춰야 합니다.'Debugging model'이 COFF 일 경우, 버전 9 부터 변수 관련 성능 개선 및 버그 수정이 제한됩니다.
이후 MCU 프로그램의 링크 출력 파일 (확장명 out, elf, axf, x 등)을 선정합니다. easyDSP 프로젝트 생성 전에 링크 출력 파일이 존재해야 함에 유의 바랍니다.
또한 TI 사 C28x 을 제외하고는 모든 다른 MCU 에 대해서 출력 파일은 DWARF 형식 디버깅 정보가 포함된 파일이어야 합니다.
한번 프로젝트가 생성되면 Basic 페이지의 속성은 수정이 불가합니다. 수정이 필요할 경우 새로 프로젝트를 생성해야 합니다.



멀티 코어 MCU 의 경우, 추가 주의 사항이 있습니다. 먼저 실제 동작하는 코어의 출력 파일을 모두 지정해야 합니다. easyDSP 는 지정된 출력파일을 사용해 램 부팅 또는 플래시 프로그래밍을 진행합니다. 또한 easyDSP 가 실제 통신하는 코어를 'Communication with easyDSP' 체크박스에 지정합니다. 하기 그림에서 easyDSP 는 CPU1, CPU2 코어와 통신하며, 실제 MCU 는 4 개의 코어를 동작시키는 경우입니다.

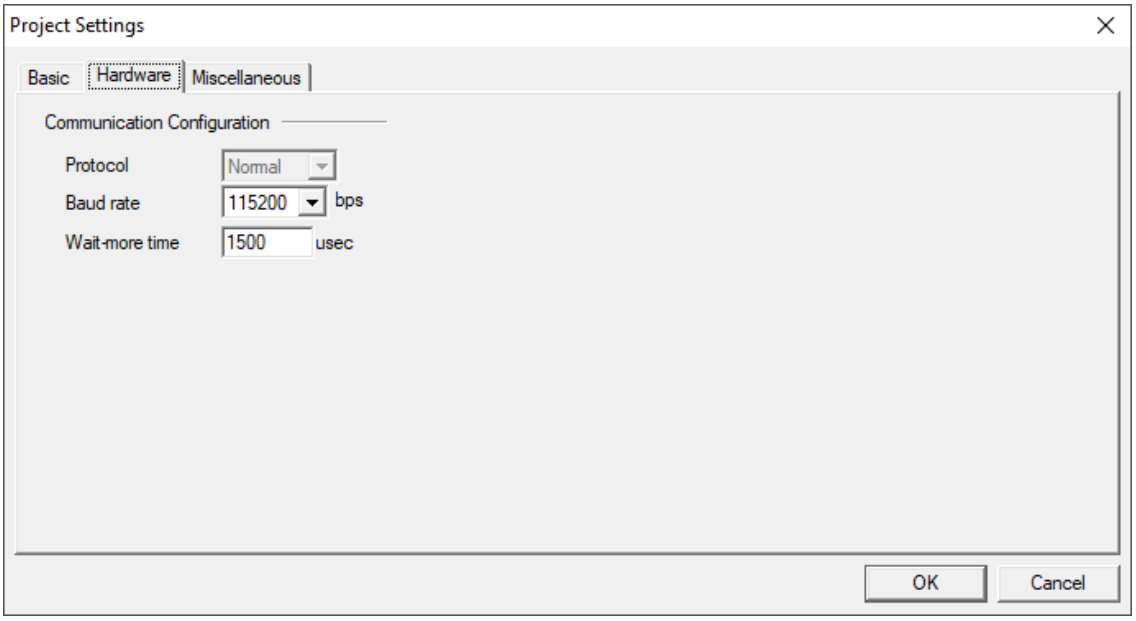


'Hardware' 속성 페이지에서는 통신을 위한 각종 하드웨어의 성질을 설정합니다.

Protocol : 사용자 선택이 불가능합니다.

Baud rate : SCI/UART 통신에 사용되는 보드레이트를 설정합니다. 이 값은 MCU 의 SCI/USART 통신 보드레이트와 동일해야 합니다.

Wait-more time : 통신시 MCU 의 반응을 기다리는 추가 시간입니다. easyDSP 는 통신시 적절한 시간 동안 MCU 의 반응을 기다립니다. 만약 사용자의 MCU 프로그램의 수행 시간이 길 경우, easyDSP 와의 통신을 위한 인터럽트 루틴의 호출이 지연될 수 있습니다. 이 경우, easyDSP 가 기다리는 시간을 늘려줘야 적절히 통신이 수행될 것입니다. 최대 30msec 까지 설정 가능하며 보통 1msec 를 권장합니다.



'Miscellaneous'속성 페이지에서는 기타 설정을 선정합니다.

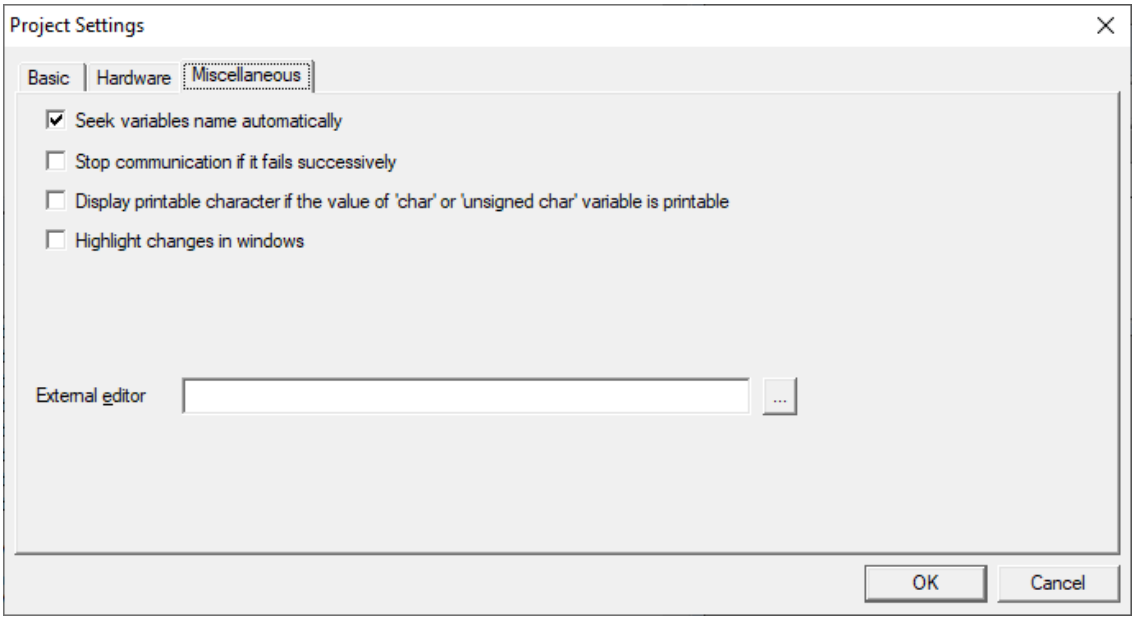
'Seek ...' 기능은 각종 윈도우에서 사용자가 변수명을 타이핑할 때 easyDSP 가 적절한 변수명을 자동으로 찾아주는 기능이며,

'Stop...'기능은 일정 횟수 이상 통신 실패시 easyDSP 가 자동적으로 통신을 중단하는 기능입니다.

'Display printable ...' 는 char 또는 unsigned char 로 설정된 변수가 문자로 표현될 수 있을 때 (변수값이 0x20~0x7F 사이), 문자로 표시하게 됩니다.

'Highlight changes ..'는 변경된 변수값을 노란 배경색으로 표시합니다.

External editor : Tools > Editor 메뉴에서 호출할 에디터를 설정합니다.



Open 메뉴:

기존의 프로젝트를 엽니다.

Set & Save 메뉴 :

현재 프로젝트의 성질을 선정합니다. 사용방법은 'New' 메뉴와 동일합니다.

Close 메뉴:

현재의 프로젝트를 닫습니다.

Delete 메뉴 :

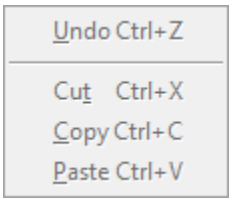
easyDSP 는 프로젝트 환경 저장을 위해 몇 가지 파일들을 프로젝트 폴더 또는 컴파일러 출력 파일 폴더안에 생성합니다. 이들은 다음과 같습니다.

MCU	easyDSP 프로젝트 폴더	컴파일러 출력 파일이 위치한 폴더
공통	프로젝트 이름.ezd : 프로젝트 환경 저장 프로젝트 이름.vars : 변수 정보 저장 프로젝트 이름.cfg : 기타 정보 저장	
C28x	easyDSP_FlashApiWrapper.out easyDSP_FlashApiWrapper.ou~ easyDSP_FlashApiWrapper.ez.bin : 플래시 동작을 위해 easyDSP 가 생성한 임시 파일들	출력파일 이름.ez.bin : 램부팅 및 플래시 프로그래밍 파일 (Gen2) 출력파일 이름.ez.hex : 플래시 프로그래밍 파일 (Gen3+)
PSOC		출력파일 이름.ez.cyacd : 플래시 프로그래밍 파일
STM32 TM4C MSPM0 RA/RX PSOC XMC TX(Z) LPC S32		출력파일 이름.ez.hex : 램부팅(지원된다면) 및 플래시 프로그래밍 파일
AM2x		출력파일 이름.ez.appimage : 램부팅 및 플래시 프로그래밍 파일

Delete 메뉴는 선택한 프로젝트에 관련된 모든 easyDSP 관련 파일을 모두 지웁니다.

8.2 Edit

Edit 메뉴

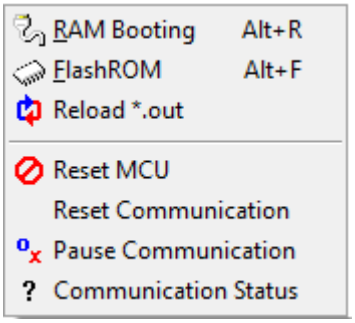


설명의 필요가 있나요? ^^;

8.3 MCU

8.3.1 공통

MCU 메뉴



RAM Booting 메뉴

Flash ROM 메뉴

해당 MCU 메뉴를 참조하세요.

C28x

STM32

S32

AM263x

TM4C

MSPM0

PSoC4

XMC1

XMC4

RA

RX

TX, TXZ3

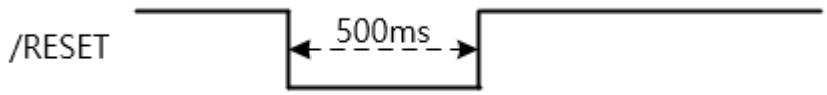
LPC

Reload *.out 메뉴

프로젝트에서 설정된 컴파일러 출력 파일 (예를 들어 *.out, *.elf, *.axf)을 재로딩합니다. 디버거와 easyDSP 를 같이 사용하여 개발하시거나, easyDSP 를 통신만을 위해 사용할 경우 (즉, /RESET, BOOT 핀 연결없이) 유용합니다.
XMC1 시리즈처럼 플래시 프로그래밍이 지원되지 않는 MCU 의 경우, MCU 가 디버거 등 다른 툴로 프로그래밍될 때마다, 본 버튼을 클릭하여 재프로그래밍된 변수 정보를 업데이트해야 합니다.

Reset MCU 메뉴

easyDSP 포드의 /RESET 단자가 500ms 동안 Low 로 내려가며, MCU 를 하드웨어 리셋하여 플래시로 부팅하게 합니다. 이 때 easyDSP 포드의 /BOOT, BOOT 단자는 신호를 출력하지 않습니다.
단 XMC1 시리즈처럼 플래시 프로그래밍이 지원되지 않는 MCU 의 경우, 본 기능은 지원되지 않습니다.



Reset Communication 메뉴

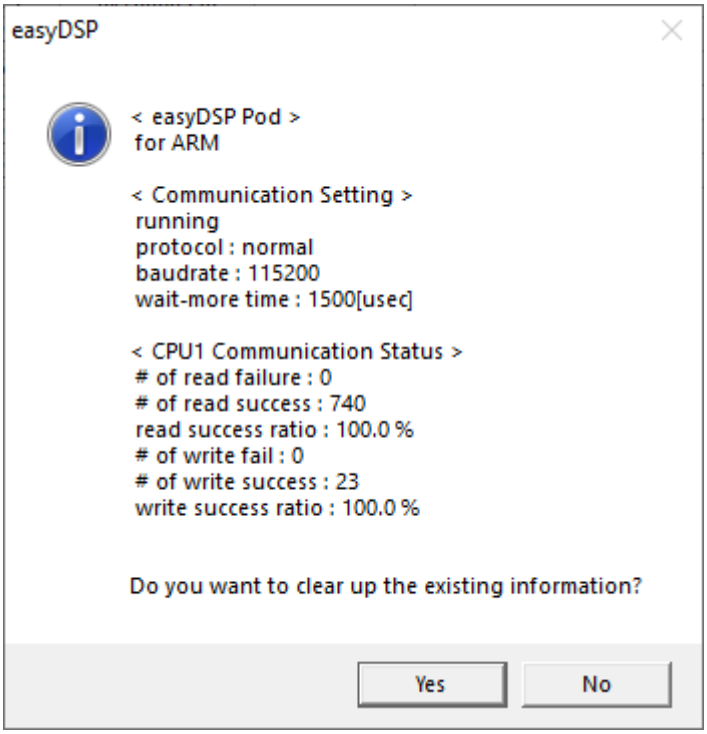
MCU 내 easyDSP 용 인터럽트 처리 루틴의 통신 시퀀스를 초기화합니다.

Pause Communication 메뉴

easyDSP 의 통신을 멈춥니다. 이 메뉴는 'Resume Communication'메뉴와 토글됩니다.

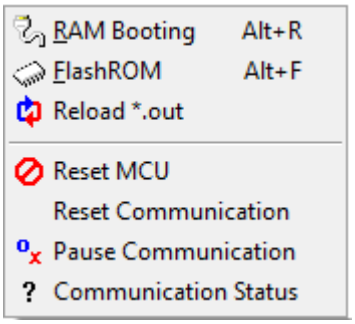
Communication Status 메뉴

easyDSP pod 타켓 MCU 및 현재 통신 상태(전송 실패율 등)을 표시합니다. 통신환경을 안정적으로 구축하여 성공률(success ratio)이 90% 이상이어야 작업상 무리가 없습니다.



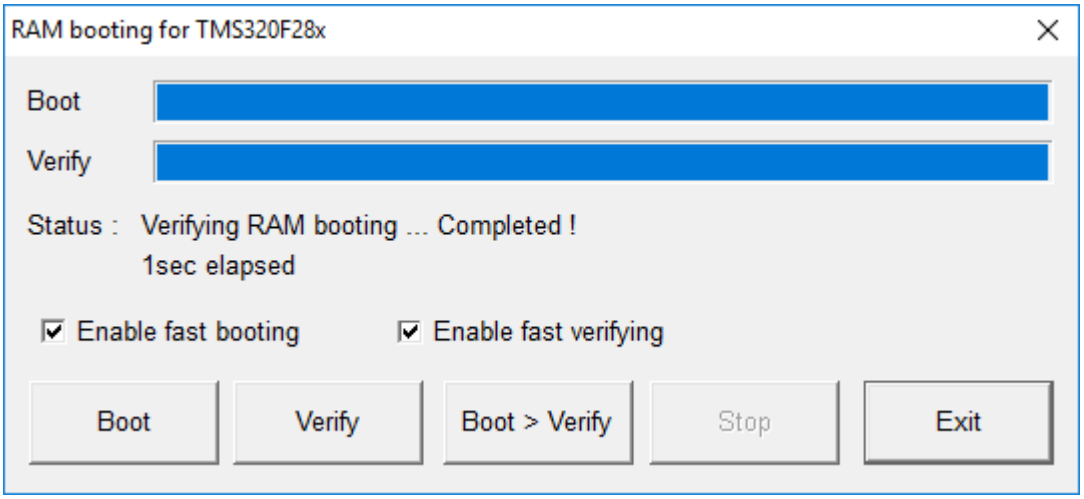
8.3.2 C28x

MCU 메뉴 (TI C28x)



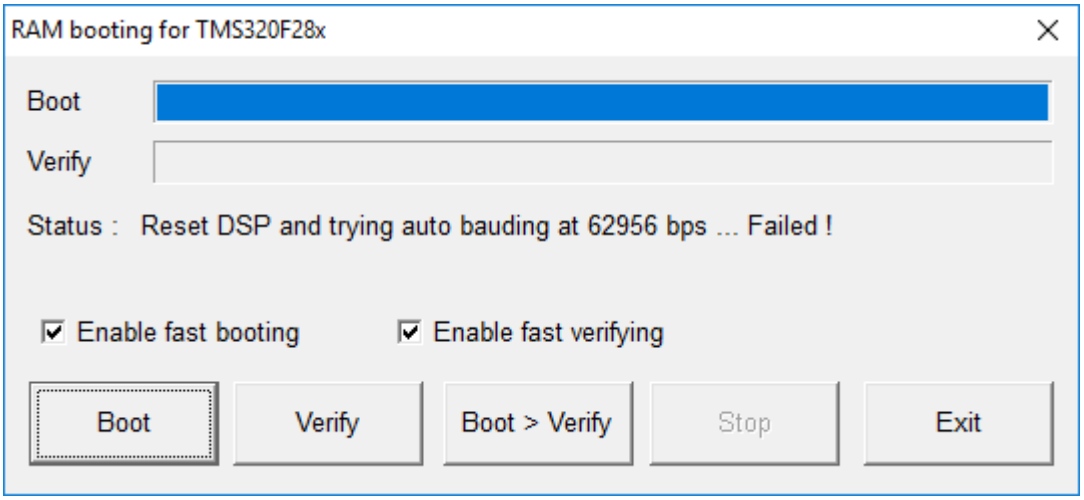
RAM Booting 메뉴

사용자 프로그램을 RAM 으로 부팅합니다. 플래시는 사용되지 않습니다. easyDSP 의 모니터링 기능은 정지되며, 다음과 같은 대화상자가 나타납니다.



'Boot'버튼을 누르면 RAM 부팅을 시작합니다. 먼저 사용자 프로그램이 RAM 부팅에 적합한지를 점검합니다. 본 점검을 통과했을 경우에만 RAM 부팅을 시작합니다. 만약 사용자 프로그램이 갱신되었다면 갱신된 프로그램으로 부팅할지를 자동 확인합니다.
'Enable fast booting' 또는 'Enable fast verifying' 체크박스를 선택하면 좀 더 빠른 속도로 해당 작업을 수행하게 됩니다.
'Enable fast verifying'의 경우 빠른 통신을 위한 리소스가 충분치 않은 경우 동작이 원활하지 않을 수 있습니다. 이 경우 비활성화하여 주세요.

부팅 실패시 나타날 수 있는 메시지에 대해서 설명합니다.



본 메시지는 DSP 가 RAM 부팅에 진입되지 않는 경우를 나타내며, 보통 하드웨어 결선이 올바르지 못할 경우 발생합니다.

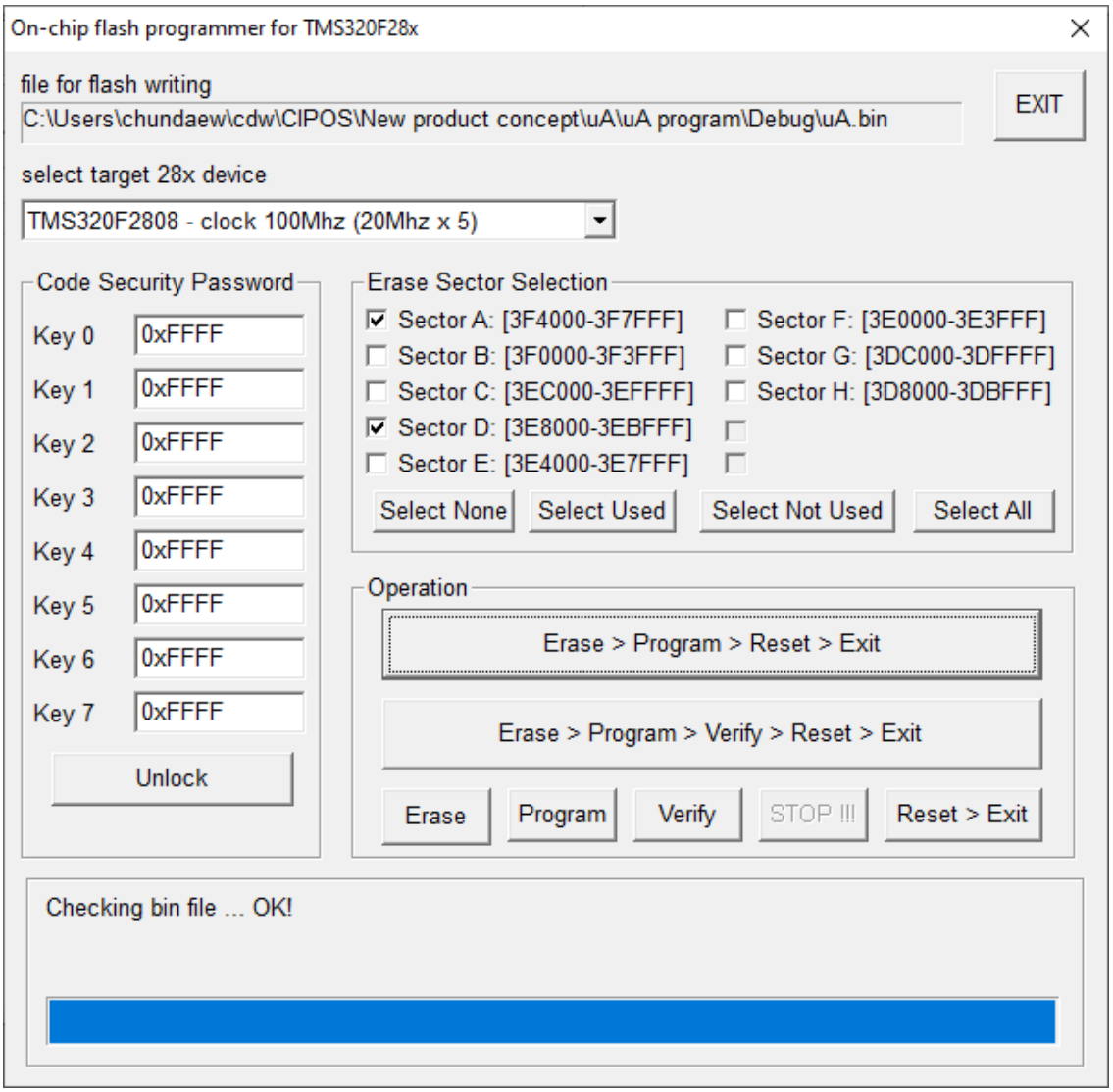
'Verify'버튼은 부팅 후 부팅 데이타 전송이 제대로 수행되었는지를 확인합니다. 만약 'Verify'실패 시에는 아래와 같은 메시지가 나오며 그 의미는 다음과 같습니다.

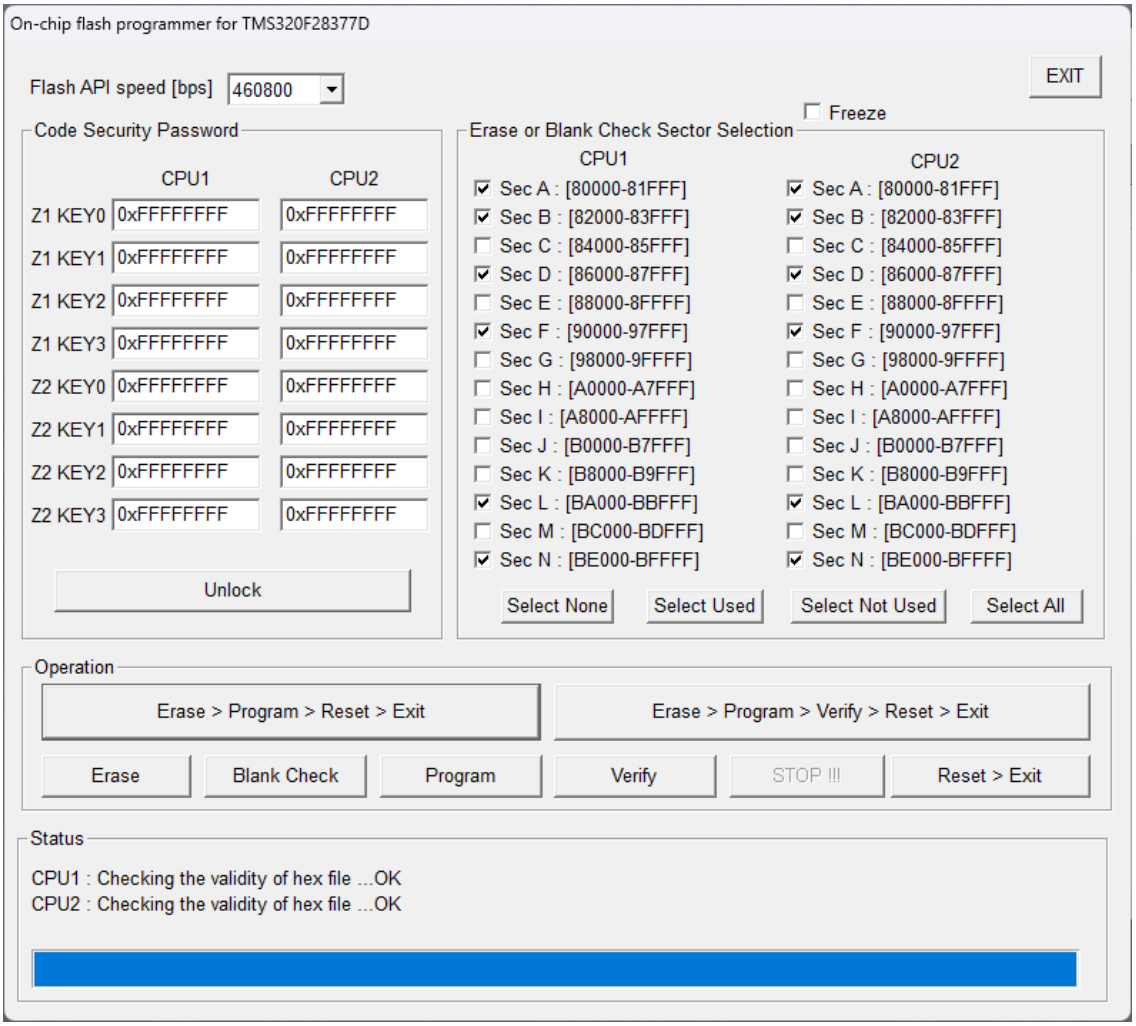
메모리 번지 0x240000 에서, 램 부팅한 데이타는 0x28AD 이지만 현재 읽혀진 값은 0x159D 이다.

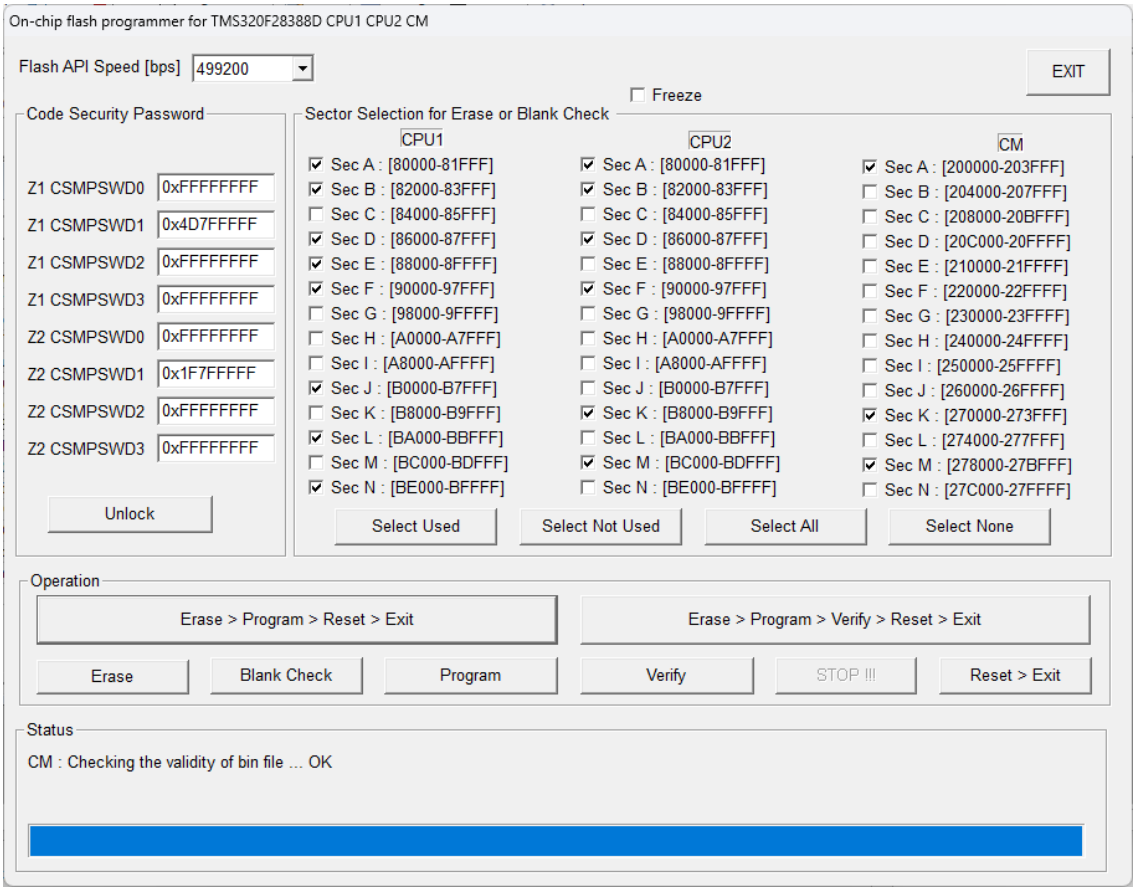
'Boot > Verify' 버튼은 'Boot' 버튼과 'Verify' 버튼의 기능을 동시에 수행합니다.
'Stop' 버튼을 누르면 현재 동작이 중지됩니다.

Flash ROM 메뉴 (Gen2 MCU, F2837xD 및 F2837x)

사용자 프로그램을 플래시에 프로그래밍합니다.
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.




On-chip flash programmer for TMS320F28377D
Flash API speed [bps]
460800
EXIT
Freeze
Code Security Password
CPU1
CPU2
Z1 KEY0
Z1 KEY1
Z1 KEY2
Z1 KEY3
Z2 KEY0
Z2 KEY1
Z2 KEY2
Z2 KEY3
0xFFFFFFFF
Unlock
Erase or Blank Check Sector Selection
Sec A : [80000-81FFF]
Sec B : [82000-83FFF]
Sec C : [84000-85FFF]
Sec D : [86000-87FFF]
Sec E : [88000-8FFFF]
Sec F : [90000-97FFF]
Sec G : [98000-9FFFF]
Sec H : [A0000-A7FFF]
Sec I : [A8000-AFFFF]
Sec J : [B0000-B7FFF]
Sec K : [B8000-B9FFF]
Sec L : [BA000-BBFFF]
Sec M : [BC000-BDFFF]
Sec N : [BE000-BFFFF]
Select None
Select Used
Select Not Used
Select All
Operation
Erase > Program > Reset > Exit
Erase > Program > Verify > Reset > Exit
Erase
Blank Check
Program
Verify
STOP !!!
Reset > Exit
Status
CPU1 : Checking the validity of hex file ...OK
CPU2 : Checking the validity of hex file ...OK



동작 순서는 다음과 같습니다.

단계 1 : 사용 MCU 및 클럭 구성에 따라 적절하게 타켓 소자를 선정합니다. 일부 MCU 의 경우에만 해당 메뉴가 존재합니다.

단계 2 : 버튼 또는 체크박스를 사용하여 Erase 및 Blank Check 의 대상이 되는 sector 를 선정합니다.

Select Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택하며 Select Not Used 버튼은 그 반대입니다.

일부 MCU 의 경우 Freeze 체크 박스를 선택하여 섹터 선택을 비활성화시킬 수 있습니다.

단계 3 : 사용자가 Erase, Blank Check, Program, Verify, Unlock 버튼을 처음 누를 때에는, DSP 는 플래쉬롬을 처리하기 위한 프로그램 (주의 : 사용자 프로그램 아님)으로 램 부팅합니다.

만약 컴파일러 출력 파일이 갱신되었다면 갱신된 출력 파일을 사용할 것인지 사용자에게 확인합니다.

한번 클릭으로 모든 동작을 수행하는 것도 가능합니다 (예 : Erase > Program > Reset > Exit)

단계 4 : 이후 통신을 통해 원하는 동작을 수행합니다.

단계 5 : 플래시 롬 대화상자를 나갈 때, 'Reset > Exit' 버튼을 클릭하여 MCU 리셋을 수행합니다. MCU 는 플래시 롬 모드로 부팅하게 되며 이로서 사용자 프로그램이 수행됩니다.

주의) MCU 종류에 따라 상기 대화 상자의 형식이 다를 수 있습니다.

주의) 2837xD, 2838xS(D)에서 데이타를 프로그램할 때, 자동 생성된 ECC(Error Correction Code)가 함께 사용됩니다.

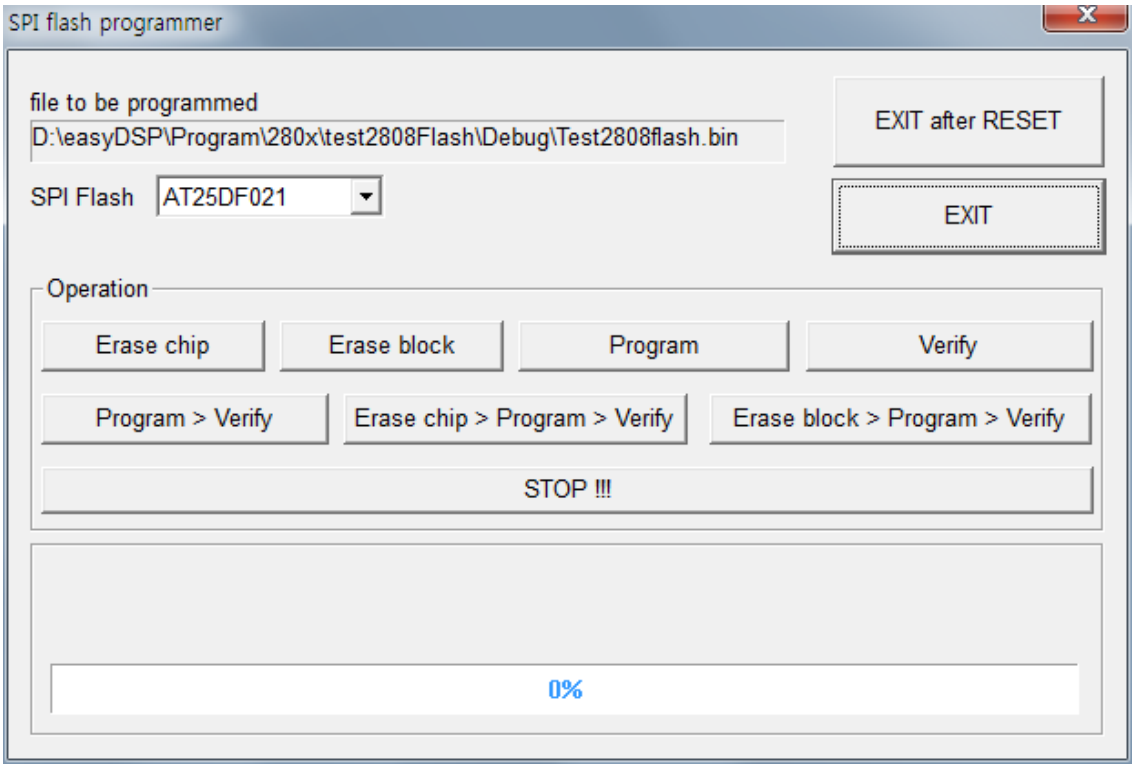
주의) 아래 메뉴가 활성화 되어 있는 경우, FlashAPI wrapper 의 통신 속도를 조절하여 시간을 단축할 수 있습니다. 단, 특정 bps 는 동작하지 않을 수 있습니다.

여기서 bps 값은 변수 모니터링에 사용되는 bps 값과 무관하기에, easyDSP 소스 파일의 헤더 파일이나,

프로젝트 세팅에서 선정하는 bps 와 매칭하지 말아야 합니다.

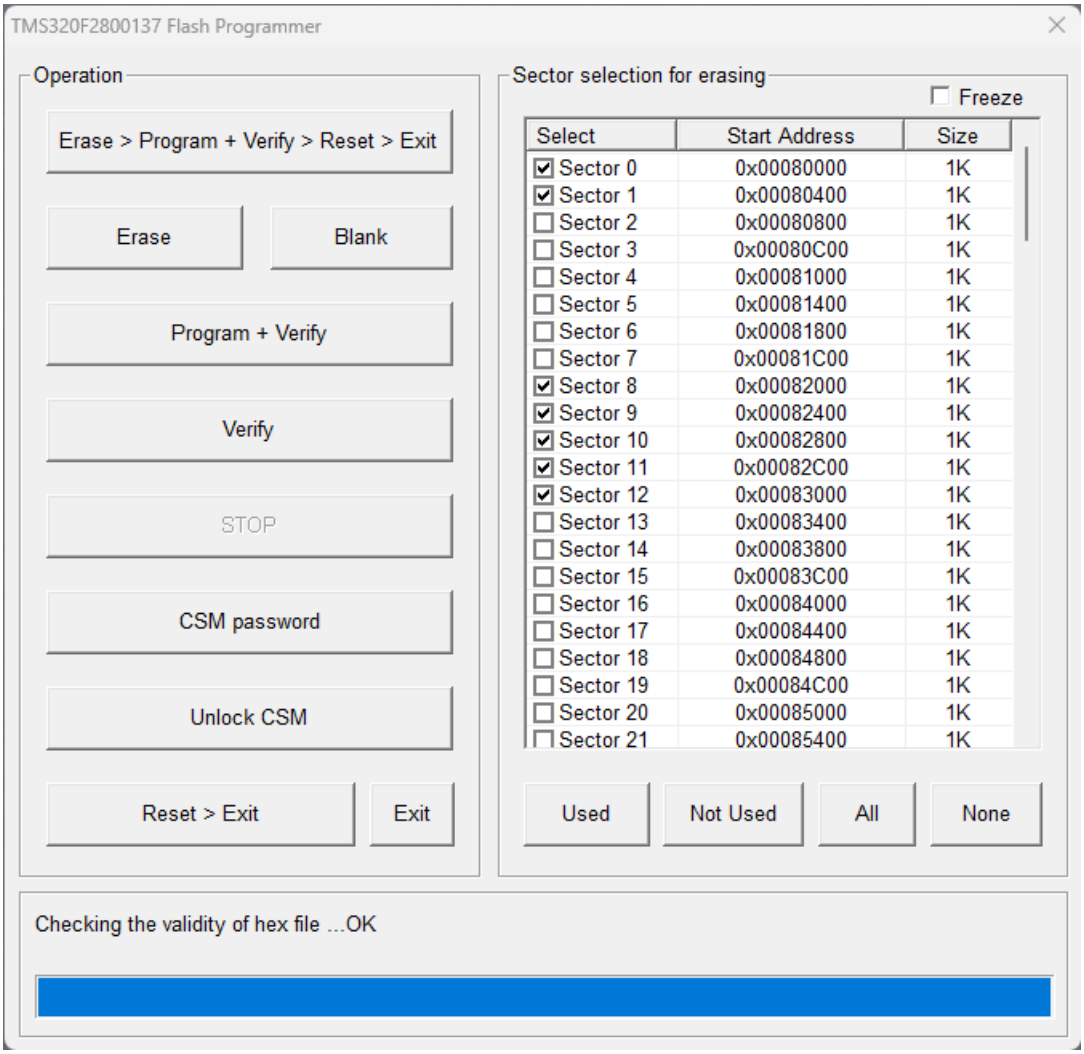
Flash ROM 메뉴 (C2834x 계열)

2834x 의 경우에는 내부 플래시 롬이 없으므로, 외부 플래쉬 롬을 지원합니다. 현재 지원되는 플래시 롬은, ATMEL 사의 SPI 통신용 플래시 롬 AT25DF021(2M bit), AT25DF041(4M bit), AT26DF081(8M bit), AT25DF321(32M bit) 및 Numonyx 사의 M25P20(2M bit), M25P40(4M bit), M25P80(8M bit), M25P16(16M bit), M25P32(32M bit)입니다. easyDSP 는 사용자 프로그램을 SPI-A 모드로 부팅할 수 있는 형식에 맞춰 플래시 롬을 라이팅합니다. 두 가지 방식의 Erase 를 지원하며, 전체 플래시 롬을 지우는 'Erase chip' 및 프로그램 라이팅에 사용된 영역만을 지우는 'Erase block'이 있습니다. 'Erase block'시에는 최소 4K bytes 영역으로 지우게 되므로, 사용자 프로그램이 실제로 라이팅된 영역보다 보통 더 큰 영역(최대 4K bytes)을 지우게 됩니다. ATMEL 사의 제품은 Sector Protection 기능이 지원되는데, easyDSP 에서는 모든 섹터를 protection 하지 않는 Global Unprotect 를 사용함에 주의하십시오. 또한 SPI-A 부팅모드에서는 부팅 속도를 조절하기 위해 LOSPCP 값 및 SPIBRR 값을 선정하게 되어 있는데, easyDSP 는 이를 LOSPCP=2, SPIBRR= 0 으로 고정하여 사용합니다.



Flash ROM 메뉴 (나머지 계열)

Gen3 싱글코어 및 멀티코어 (예 : F28Px) 제품을 지원하기 위한 메뉴입니다.
사용자 프로그램을 플래시에 프로그래밍합니다.
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



사용 순서는 다음과 같습니다.

단계 1 : 필요시 CSM password 버튼을 눌러 CSM 키값을 입력하고, Unlock CSM 버튼을 눌러 CSM 을 해제합니다.

단계 2 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용).

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

각 섹터의 체크 박스를 일일이 클릭하여 섹터별로 선택할 수도 있습니다.

Freeze 체크 박스를 선택하면 섹터 선택을 비활성화시킬 수 있습니다.

단계 3 : Erase, Blank, Program+Verify, Verify 버튼을 처음 사용시 MCU 리셋이 걸리고 MCU 는 부트로더로 진입하게 합니다.

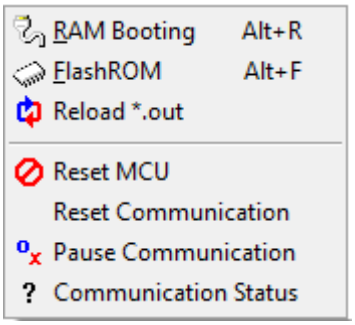
단계 4 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

데이타를 프로그램할 때, 자동 생성된 ECC(Error Correction Code)가 함께 사용됩니다.

단계 5 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 부트로더 프로그램이 지속 수행됨에 유의하세요.

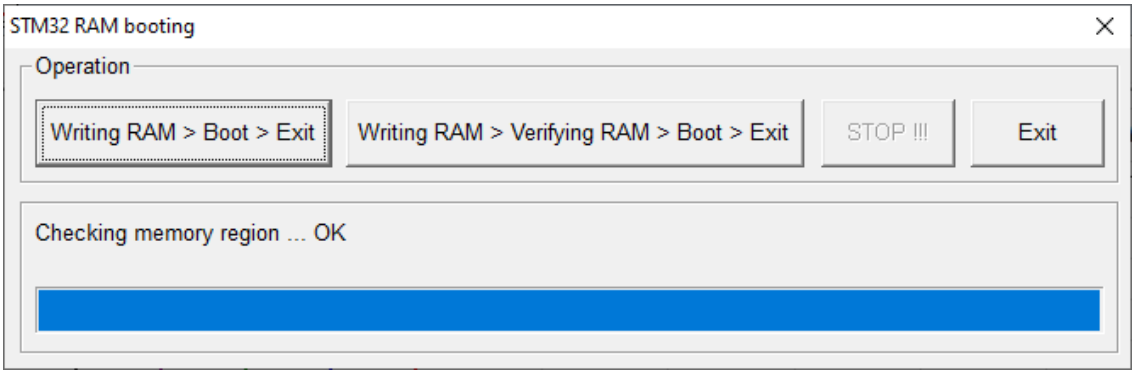
8.3.3 STM32

MCU 메뉴 (ST STM32)

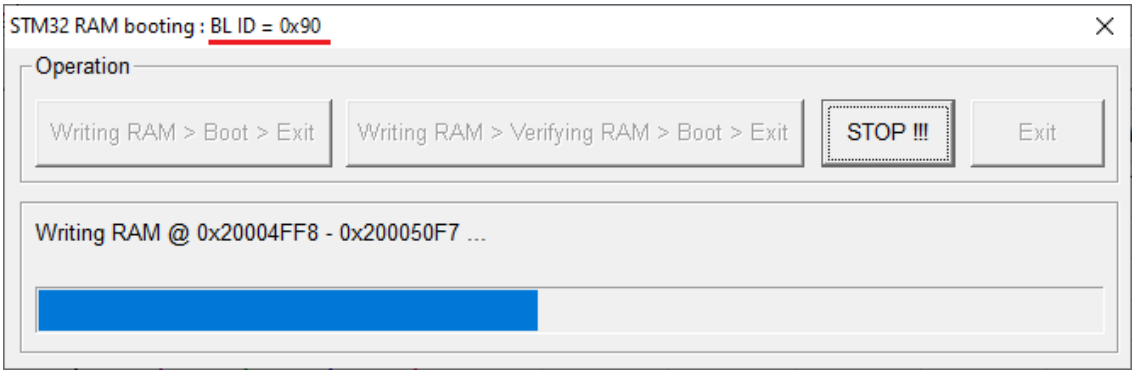


RAM Booting 메뉴

사용자 프로그램을 RAM 으로 부팅합니다. 플래시는 사용되지 않습니다.
Dual core MCU 경우 램부팅 지원되지 않음에 유의하세요.
easyDSP 의 모니터링 기능은 정지되며, 다음과 같은 대화상자가 나타납니다.



먼저 컴파일러 출력 파일을 검토하여 사용되는 메모리 영역이 RAM 부팅에 적합한지 확인합니다. 만약 플래시 영역이 발견된다면 에러 메시지를 표시합니다.
해당 버튼을 누르면 RAM 부팅을 시작합니다. 만약 사용자 프로그램이 갱신되었다면 갱신된 프로그램으로 부팅할지를 자동 확인합니다.
동작하기 전 MCU 에 내장된 부트로더의 버전을 하기와 같이 상단에 표시합니다.
만약 램부팅이 원활하지 않으면 본 버전 기반으로 최신 버전의 AN2606(STM32 microcontroller system memory boot mode)를 참조하세요.



'Stop' 버튼을 누르면 현재 동작이 중지됩니다.

Flash ROM 메뉴

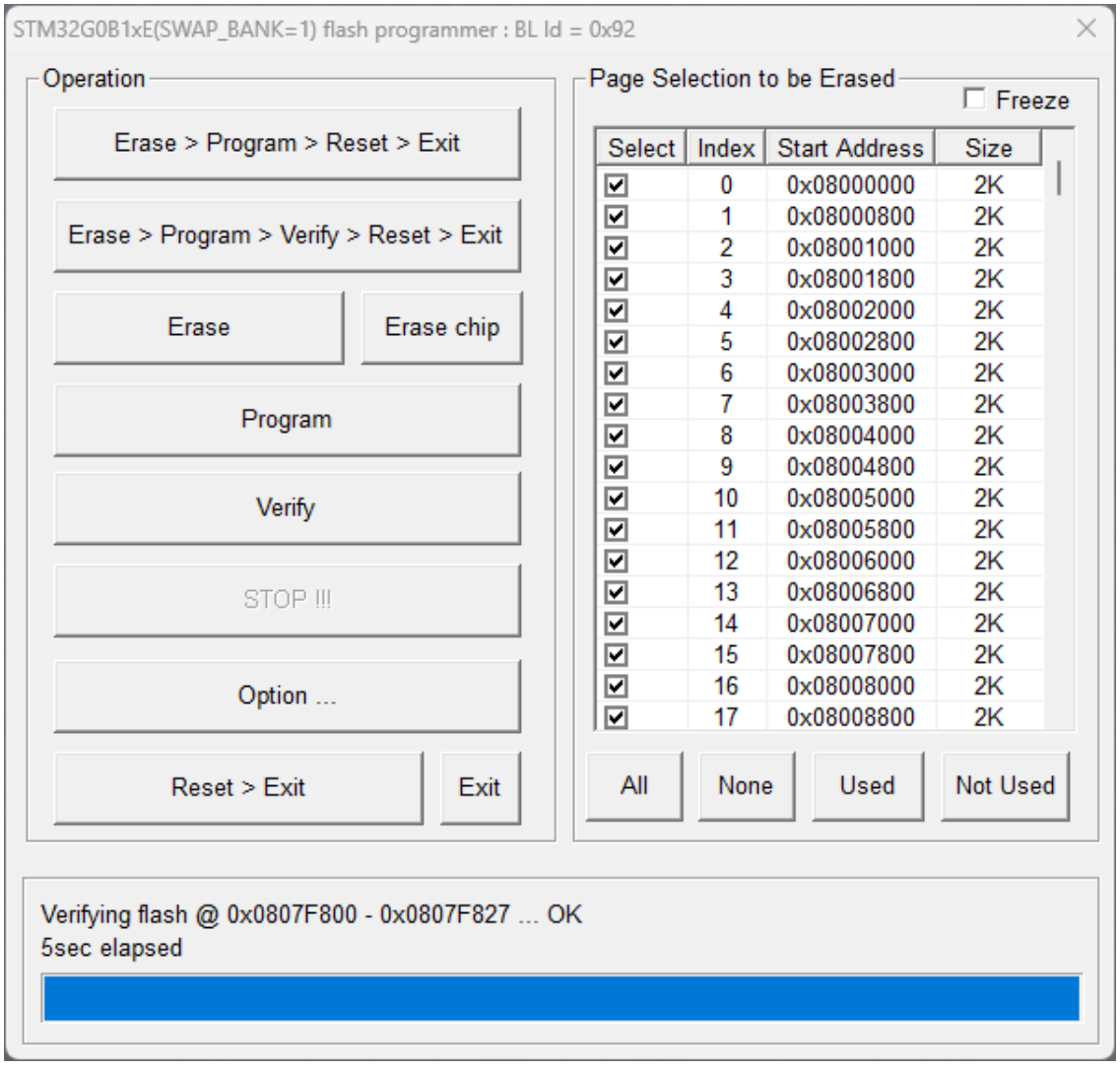
사용자 프로그램을 플래시에 프로그래밍합니다.

OTP memory, Data memory, Option byte 는 읽기 및 쓰기는 지원하지 않습니다.

보호 및 보안 기능이 활성화되어 있으면 플래시 프로그래밍이 안될 수 있습니다.

Trust Zone 이 활성화되어 있거나, Secure MPU 사용시에 기능이 제한될 수 있습니다.

easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



사용 순서는 다음과 같습니다.

단계 1 : Erase 대상이 되는 플래시 페이지를 선정합니다 (All, None, Used, Not Used 버튼 활용). 체크 박스를 사용하여 페이지별로 선택할 수도 있습니다.

Used 버튼은 사용자 프로그램이 사용하는 모든 페이지를 선택합니다. Not Used 버튼은 그 반대입니다.

Freeze 체크 박스를 선택하면 섹터 선택을 비활성화시킬 수 있습니다.

단계 2 : MCU 종류에 따라 'Option ...' 버튼이 활성화될 경우가 있습니다. 이 경우 클릭하셔서 옵션을 선택하시기 바랍니다.

예를 들어 STM32G0B1 의 경우, SWAP_BANK bit 상태를 선택합니다 (변경은 아님).

선택된 옵션은 저장되므로, 한번 선택 후 변경이 필요한 경우만 재선택하시면 됩니다.

단계 3 : Erase, Erase chip, Program, Verify 버튼을 처음 사용시 MCU 리셋이 걸리고 MCU 는 부트모드로 진입하게 합니다.

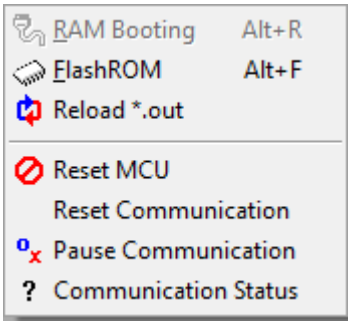
부트모드에 진입하면 Bootloader ID 가 대화상자 제목에 표기됩니다.

단계 4 : 각종 버튼을 사용하여 플래시 동작을 수행합니다. Erase chip 버튼은 페이지 선정과 상관없이 모든 플래시를 지웁니다.

단계 5 : 대화상자를 나갈 때, 'Reset > Exit' 버튼으로 MCU 를 리셋합니다. 이로서 사용자 프로그램이 수행됩니다. MCU 리셋 없이 대화상자를 나가면 부트로더 프로그램이 지속 수행됨에 유의하세요.

8.3.4 S32

MCU 메뉴 (NXP S32)



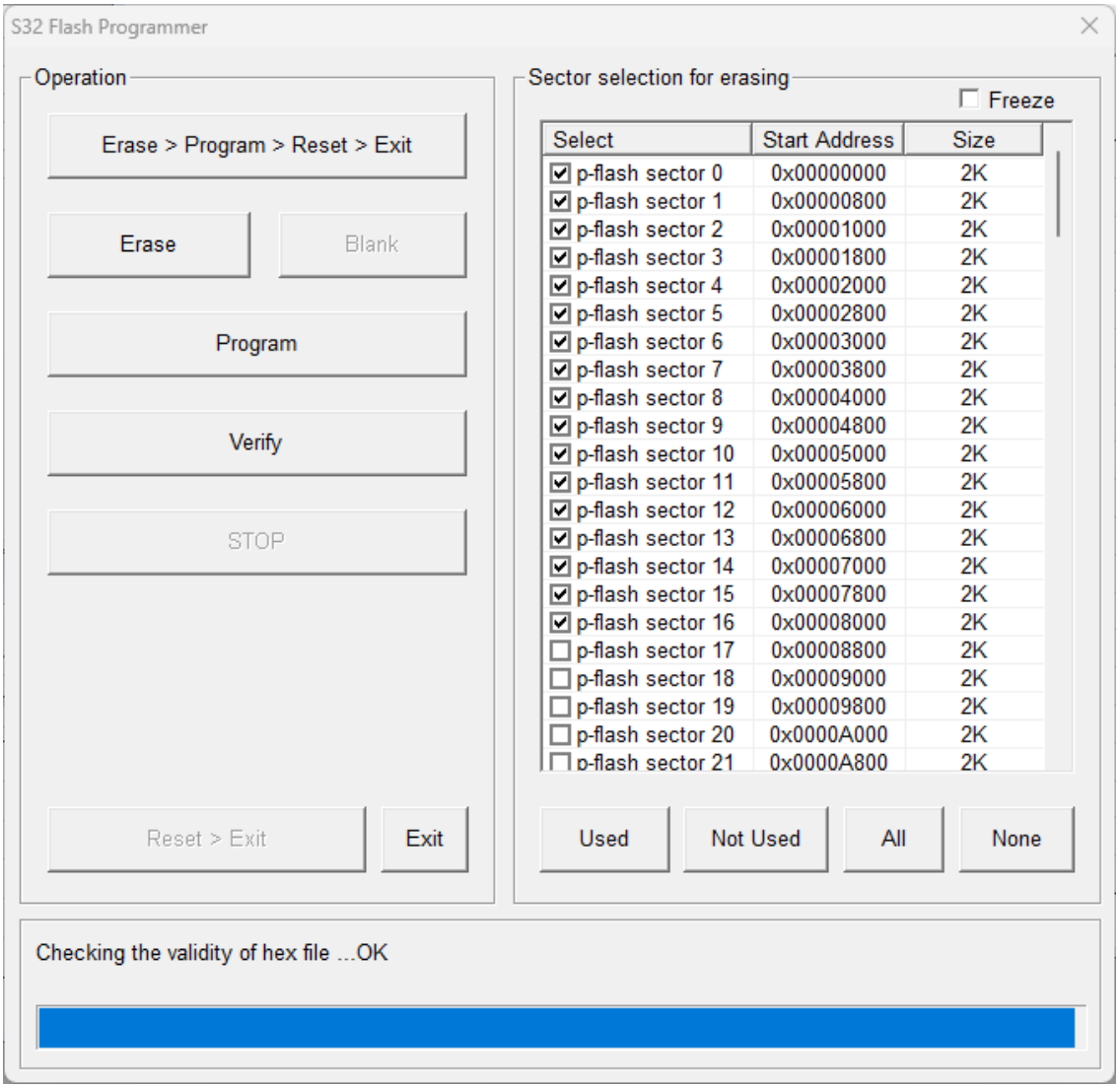
RAM Booting 메뉴

본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

주의 사항 : 본 메뉴는 easyS32**.h 에서 EZ_BOOTLOADER_USE 가 1 로 정의된 경우에만 동작합니다. 여기를 참조하세요 .

사용자 프로그램을 플래시에 프로그래밍합니다.
보호 및 보안 기능이 활성화되어 있으면 플래시 프로그래밍이 안될 수 있습니다.
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



사용 순서는 다음과 같습니다.

단계 1 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용).

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

각 섹터의 체크 박스를 일일이 클릭하여 섹터별로 선택할 수도 있습니다.

Freeze 체크 박스를 선택하면 섹터 선택을 비활성화시킬 수 있습니다.

단계 2 : Erase, Program, Verify 버튼을 처음 사용시 MCU 리셋이 걸리고 MCU 는 easyDSP 가 제공하는 부트로더(함수 easyDSP_boot())로 진입하게 합니다. 참고로 Blank 버튼은 동작하지 않습니다.

단계 3 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

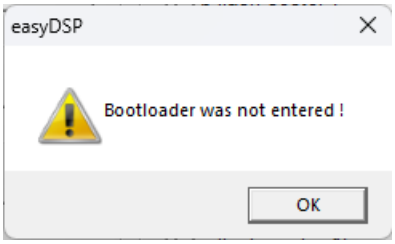
단계 4 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 부트로더 프로그램이 지속 수행됨에 유의하세요.

주의 사항 :

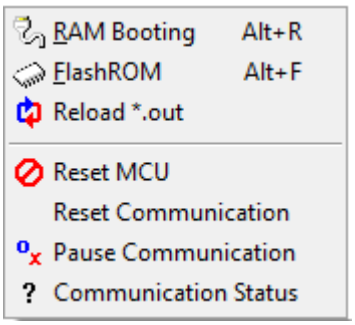
S32K MCU 는 제작업체에서 제공하는 롬부트로더가 없기에 사용자가 부트로더를 별도로 제작 사용해야 합니다. easyDSP 가 제공하는 부트로더는 사용자 프로그램 안에 함수(함수명 easyDSP_boot)로 존재합니다. 따라서 MCU 플래시에 easyDSP 부트로더가 이미 프로그래밍되어 있어야 부트로더가 작동할 수 있고 새로운 프로그램으로 플래시 재 프로그래밍할 수 있습니다.

만약 플래시가 전부 지워져 있거나, easyDSP 부트로더가 이미 플래시에 프로그래밍이 되어 있지 않다면, 플래시 프로그래밍이 지원되지 않으며 하기와 같은 메시지가 송출되며, 이 경우 디버거를 사용한 플래시 프로그래밍이 필요합니다.

즉, 최소 처음 한번은 디버거를 이용해서 플래시 프로그래밍을 해야만 합니다.

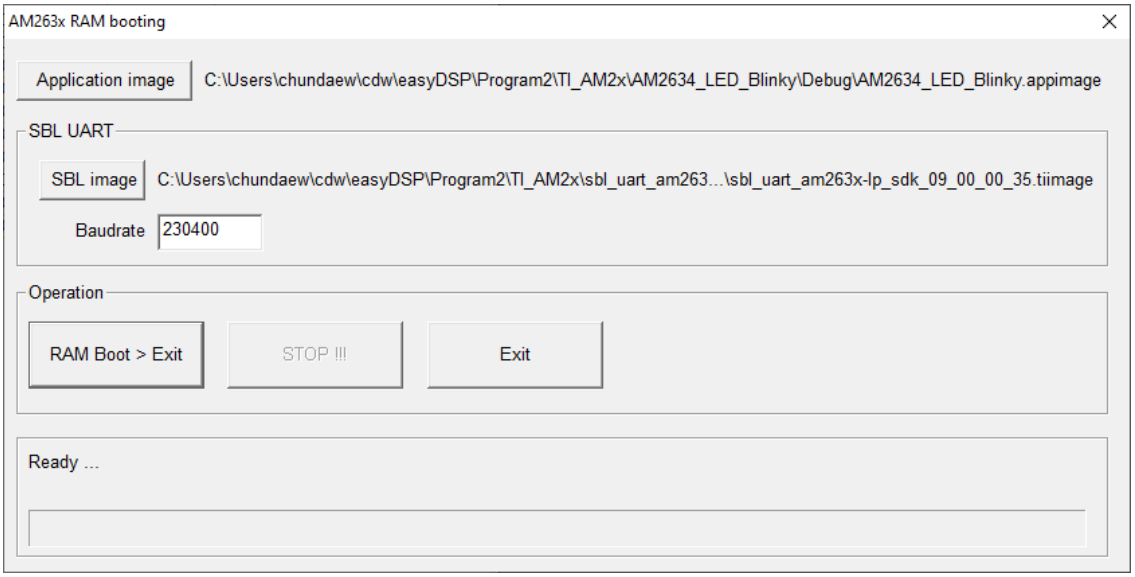


8.3.5 AM263x



RAM Booting 메뉴

easyDSP 는 TI 가 제공하는 SBL UART 기반으로 사용자 프로그램을 RAM 으로 부팅합니다. easyDSP 의 모니터링 기능은 정지되며, 다음과 같은 대화상자가 나타납니다.



easyDSP 는 별도의 SBL UART 파일 이미지를 제공하지 않습니다.
단지 사용자가 지정한 이미지 파일 (TI 제공 prebuilt SBL 또는 사용자 작성)을 다운로딩하고 이를 통해 어플리케이션이 실행될 수 있는 환경만 제공합니다.

사용 순서는 다음과 같습니다.

순서 1 : 먼저 앱 이미지 파일을 선정합니다. 기본값으로 easyDSP 프로젝트에서 사용하는 이미지파일이 자동 입력되며, 사용자가 Application image 버튼을 통해 변경 가능합니다.

순서 2 : SBL UART 버튼을 통해 SBL UART 이미지 파일을 지정하세요. 그리고 SBL UART 의 보드레이트를 입력하세요.

만약 TI 가 제공하는 prebuilt SBL (예를 들어 C:₩ti₩mcu_plus_sdk_am263x_09_00_00_35₩tools₩boot₩sbl_prebuilt 에 위치)을 사용할 경우 보드레이트는 115200 으로 입력하시고,

사용자가 작성한 SBL UART 의 경우, 설정하신 보드레이트를 입력하세요.

순서 3 : 각종 버튼을 사용하여 필요한 작업을 수행합니다.

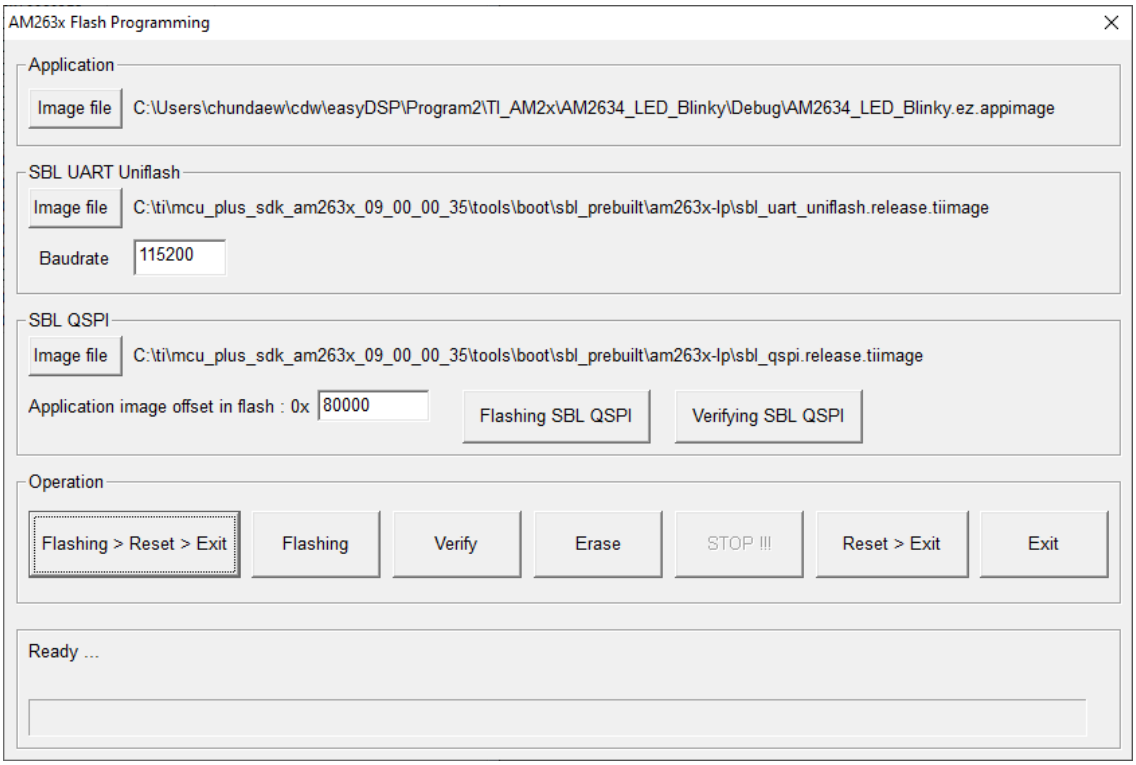
'RAM Boot > Exit' 버튼을 누르면 RAM 부팅을 시작합니다. 만약 사용자 프로그램이 갱신되었다면 갱신된 프로그램으로 부팅할지를 자동 확인합니다.

'Stop' 버튼을 누르면 현재 동작이 중지됩니다.

Flash ROM 메뉴

사용자 프로그램을 SPI 플래시에 프로그래밍합니다.

easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



easyDSP 는 별도의 SBL 파일 이미지를 제공하지 않습니다.

단지 사용자가 지정한 이미지 파일 (TI 제공 prebuilt SBL 또는 사용자 작성)을 다운로딩하고 이를 통해 어플리케이션이 실행될 수 있는 환경만 제공합니다.

사용 순서는 다음과 같습니다.

순서 1 : 먼저 앱 이미지 파일을 선정합니다. 사용자가 image file 버튼을 통해 변경 가능합니다만 기본값으로 easyDSP 프로젝트에서 사용하는 이미지파일이 자동 입력됩니다.

easyDSP 는 주어진 *rprc 파일 기반으로 앱 이미지 파일을 생성합니다 (확장자 ez.appimage).

순서 2 : SBL UART Uniflash 의 이미지 파일을 선정합니다.

TI 가 제공하는 prebuilt SBL (예를 들어 C:₩ti₩mcu_plus_sdk_am263x_09_00_00_35₩tools₩boot₩sbl_prebuilt 에 위치)을 사용할 경우 보드레이트는 115200 으로 입력하시고,

사용자가 작성한 SBL 의 경우, 설정하신 보드레이트를 입력하세요.

순서 3 : SBL QSPI 의 이미지 파일을 선정합니다.

또한 SPI 플래시내 appimage 파일이 저장될 옵셋을 지정합니다.

TI 가 제공하는 prebuilt SBL 을 사용할 경우 0x80000 을 입력하시고, 사용자가 작성한 SBL 이라면 지정하신 주소를 입력합니다.

순서 4 : 'Flashing SBL QSPI' 버튼을 클릭하여 SBL QSPI 를 SPI 플래시에 저장하세요. 한번만 저장되면 추후 재저장할 필요 없습니다.

이상의 순서 1 ~ 순서 4 는 사용자 프로그램 갱신과 상관없이 한번만 수행하면 됩니다.
사용자 프로그램을 처리하는 단계는 하기부터 입니다.

순서 5 : 'Operation' 영역내 각종 버튼을 사용하여 플래시 작업을 수행합니다.

Flashing, Verify, Erase 버튼을 처음 사용시 MCU 리셋 이후 MCU 는 부트 모드로 진입하게 되며 SBL UART Uniflash 를 로딩/실행합니다.

Flashing 은 Erase > Program > Verify 를 전부 수행하는 동작입니다. 따라서 Flashing 이전/이후에 Erase, Verify 실행은 선택 사항입니다.

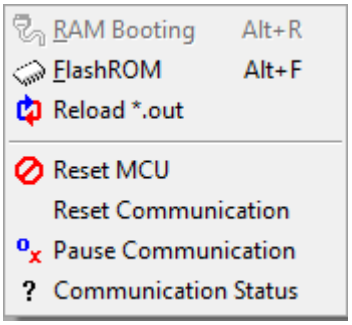
Flashing 과 Verify 는 192kB 블럭 단위로 진행됩니다.

순서 6 : 플래시 작업 이후 대화상자를 나갈 때, 'Reset > Exit' 버튼으로 MCU 를 리셋합니다. 이로서 MCU 는 QSPI (4S) - Quad Read Mode 로 진입하며 플래시에 저장된 사용자 프로그램이 실행됩니다.

주의 사항 : MCU 가 부트 모드에 진입한 이후, MCU 리셋 없이 대화상자를 나갈 경우, SBL 프로그램이 지속 실행되어 모니터링이 되지 않습니다.

8.3.6 TM4C

MCU 메뉴 (TI TM4C)

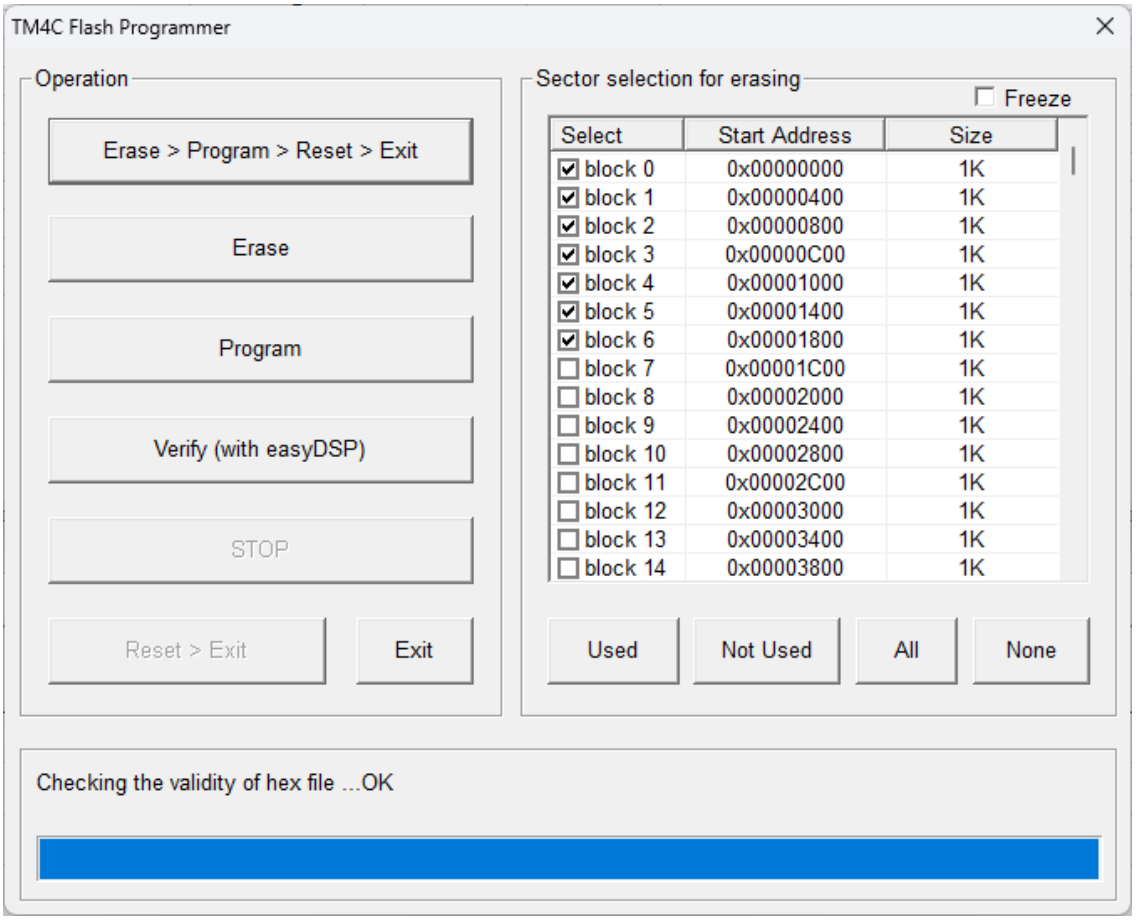


RAM Booting 메뉴

본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

사용자 프로그램을 플래시에 프로그래밍합니다.
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



사용 순서는 다음과 같습니다.

단계 1 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용).

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

각 섹터의 체크 박스를 일일이 클릭하여 섹터별로 선택할 수도 있습니다.

Freeze 체크 박스를 선택하면 섹터 선택을 비활성화시킬 수 있습니다.

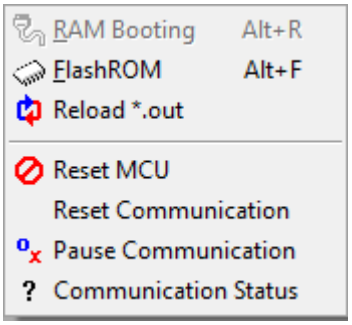
단계 2 : Erase, Program 또는 Erase > Program > Reset > Exit 버튼을 처음 사용시 MCU 리셋이 걸리고 MCU 는 롬부트로더로 진입하게 합니다.

단계 3 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

단계 4 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 롬부트로더 프로그램이 지속 수행됨에 유의하세요.

주의 사항 : MCU 롬부트로더는 Verify 기능을 제공하지 않습니다. 따라서 easyDSP 는 통신을 통한 Verify 기능을 제공하지만 이 기능은 롬부트로더 진입 이전에만 수행될 수 있습니다.

8.3.7 MSPM0



RAM Booting 메뉴

본 메뉴는 지원되지 않습니다.

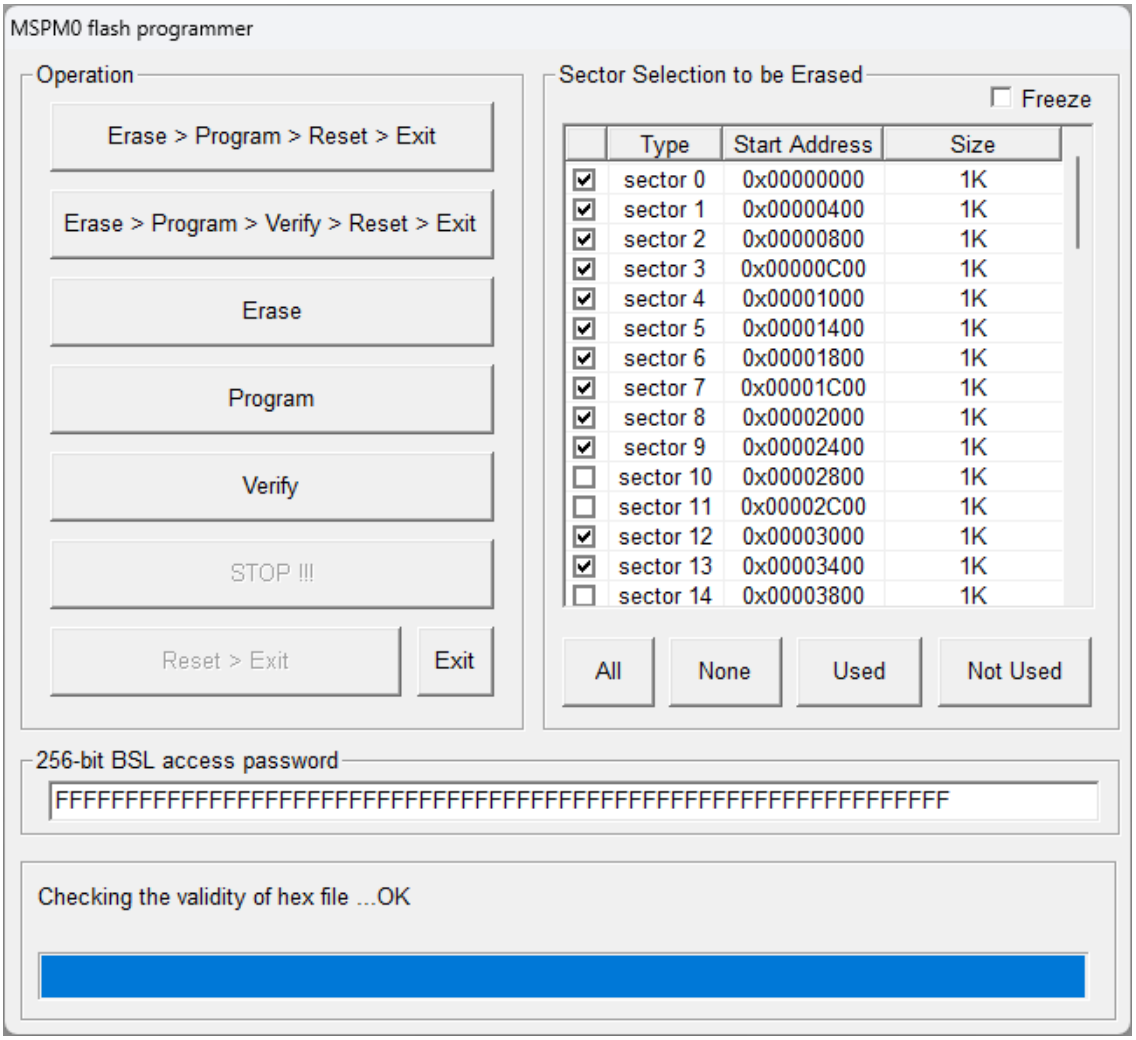
Flash ROM 메뉴

사용자 프로그램을 MAIN 플래시에 프로그래밍합니다.

단 보호 기능이 적용된 영역은 지원되지 않으니 해당 기능이 해제된 상태에서 사용하시기 바랍니다.

또한 BSL configuration 이 저장되는 NONMAIN 플래시에 대한 programming 을 지원하지 않으므로, BCR 또는 BSL configuration 을 변경할 때에는 디버거 등 다른 툴을 사용하세요.

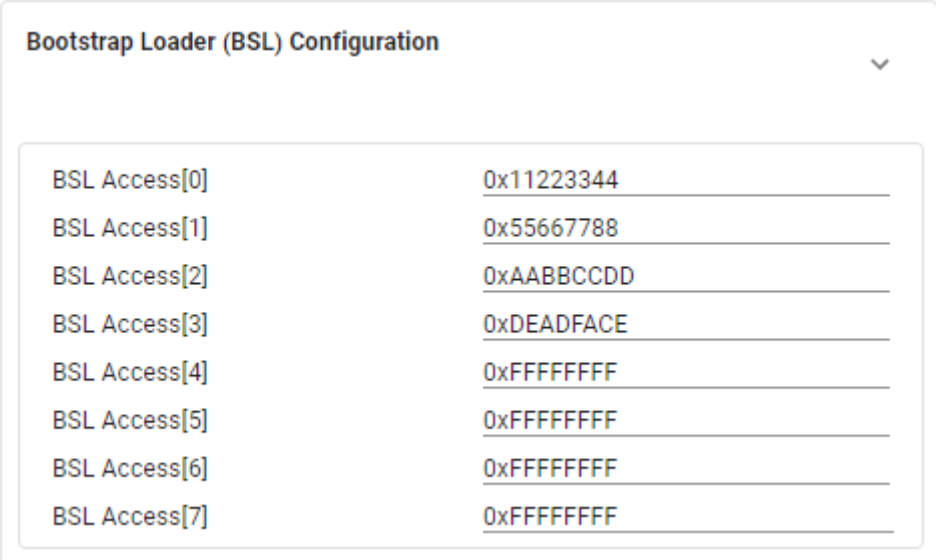
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



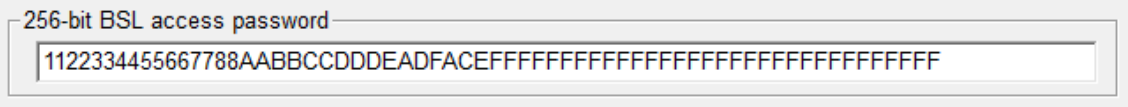
사용 순서는 다음과 같습니다.

스텝 1 : 부트모드 진입을 위한 비밀번호 32 바이트를 선정합니다. MCU 출하 초기값은 모두 0xFF 입니다.

SysConfig 에서 아래 값으로 설정할 경우,



아래와 같이 비밀번호를 입력하시기 바랍니다.



스텝 2 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용).

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

각 섹터의 체크 박스를 일일이 클릭하여 섹터별로 선택할 수도 있습니다.

Freeze 체크 박스를 선택하면 섹터 선택을 비활성화시킬 수 있습니다.

스텝 3 : Erase, Program, Verify 버튼을 처음 사용시, MCU 리셋 이후 MCU 는 부트모드로 진입하게 합니다.

스텝 4 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

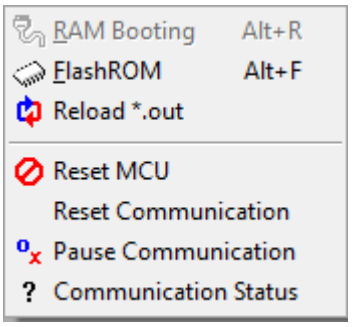
Verify 버튼의 경우, BSL Configuration 의 BSL Read Out Enable 설정에 따라 다른 동작이 수행됩니다.

만약 Read Out 이 Disabled 되어 있다면 (공장 출하 기본 설정), Verify 버튼은 플래시 메모리를 읽지 않고, 1kB 영역의 CRC 값만 확인합니다.

만약 Read Out 이 Enabled 되어 있다면, Verify 버튼은 플래시 메모리의 내용을 읽어서 바이트별로 확인합니다.

스텝 5 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 부트모드 프로그램이 지속 수행됨에 유의하세요.

8.3.8 PSoC4



RAM Booting 메뉴

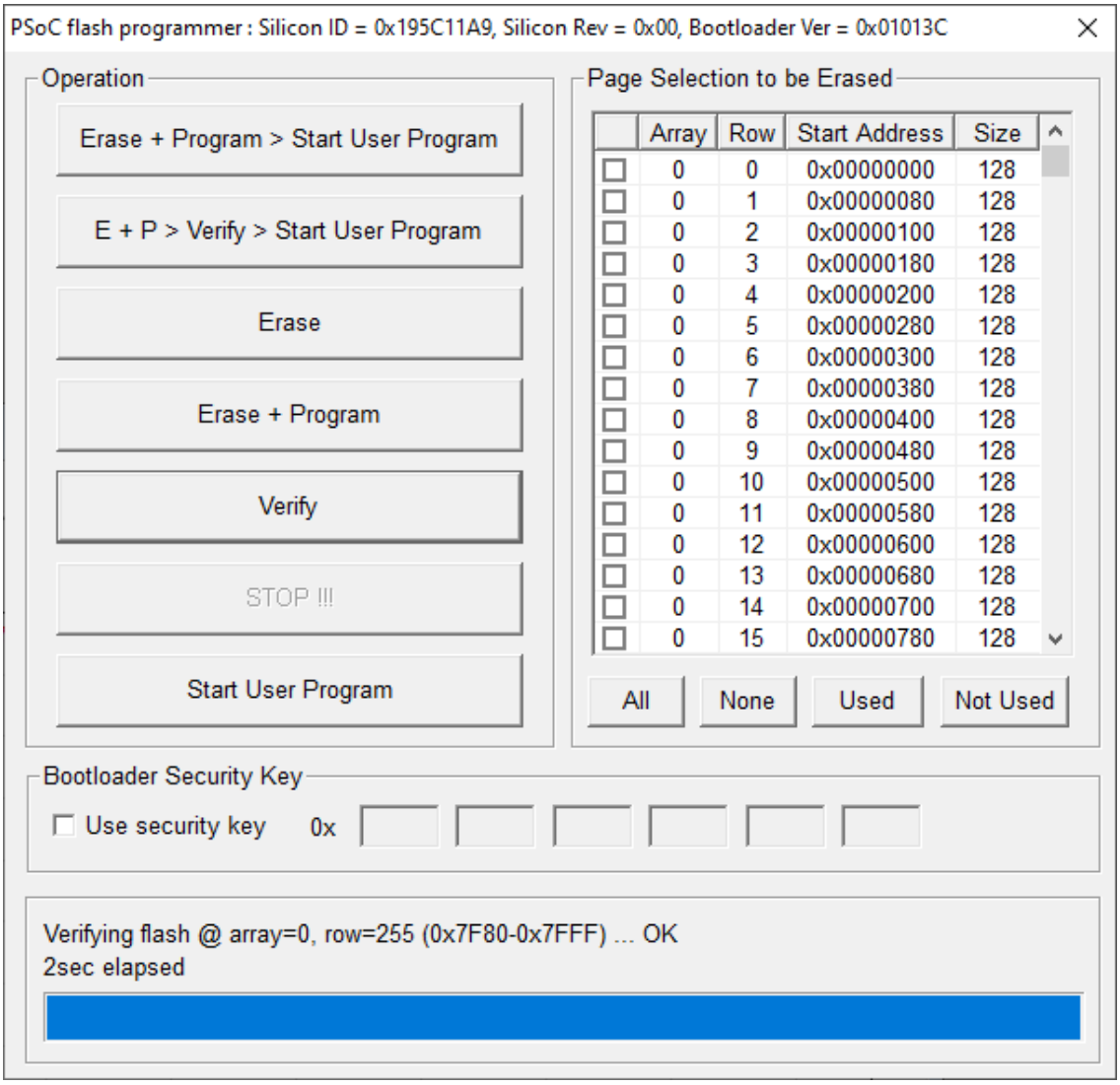
본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

사용자 프로그램을 플래시에 프로그래밍합니다.

easyDSP 는 Single-application bootloader 구성만 플래시 프로그래밍을 지원합니다.

easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



사용 순서는 다음과 같습니다.

단계 1 : Erase 대상이 되는 플래시 페이지를 선정합니다 (All, None, Used, Not Used 버튼 활용). 체크 박스를 사용하여 섹터별로 선택할 수도 있습니다.

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

단계 2 : 부트로더 보안키가 설정되었다면, Use security Key 를 체크하시고 키 입력을 합니다.

단계 3 : Erase, Erase + Program, Verify 버튼을 처음 사용시 MCU 리셋이 걸리고 MCU 는 부트로더로 진입하게 합니다.

부트로더에 진입하면 Silicon ID, Selicon Rev, Bootloader Ver 이 대화상자 제목에 표기됩니다.

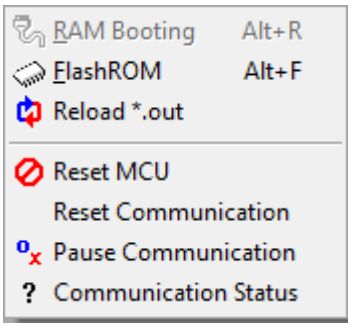
단계 4 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

단계 5 : 'Start User Probram' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다 (MCU 소프트웨어 리셋 수반). 다른 방식으로 대화상자를 나가면 부트로더 프로그램이 지속 수행됨에 유의하세요.

주의 : 부트로더 영역의 플래시 영역을 선택할 시, 해당 영역은 Erase 가 수행되지 않습니다.

8.3.9 XMC1

MCU 메뉴 (Infineon XMC1)



RAM Booting 메뉴

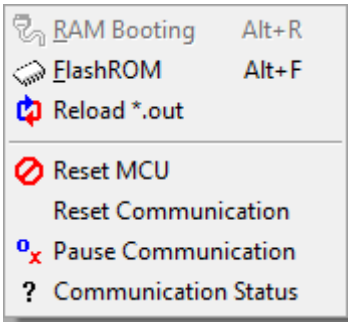
본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

본 메뉴는 지원되지 않습니다.

8.3.10 XMC4

MCU 메뉴 (Infineon XMC4)

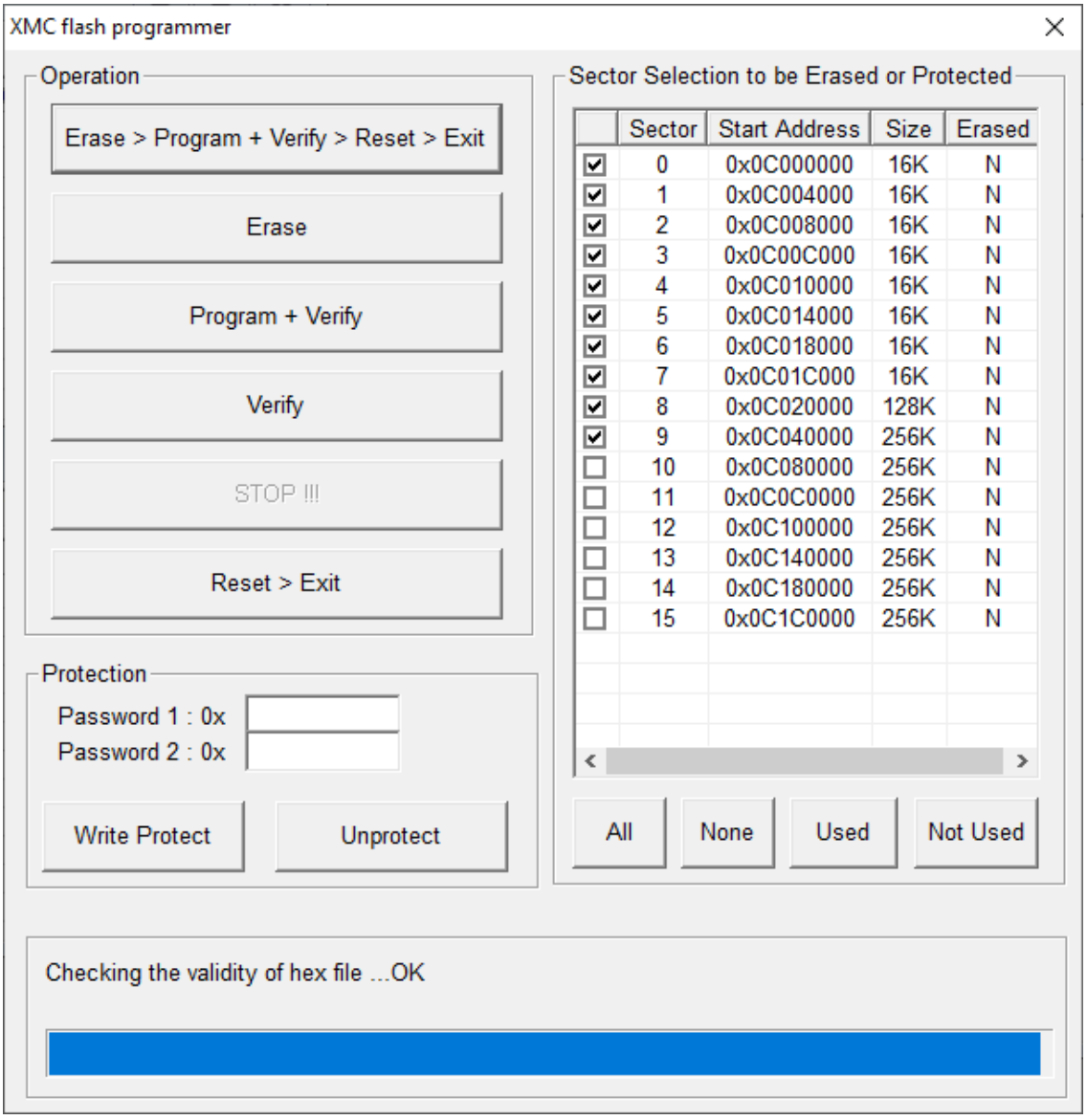


RAM Booting 메뉴

본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

사용자 프로그램을 플래시에 프로그래밍합니다.
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



사용 순서는 다음과 같습니다.

단계 1 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용). 체크 박스를 사용하여 섹터별로 선택할 수도 있습니다.

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

단계 2 : 필요시 Write protect 를 설정하실 수 있습니다.

단계 3 : Erase, Program+Verify, Verify 버튼을 처음 사용시 MCU 리셋이 걸리고 MCU 는 부트로더로 진입하게 합니다.

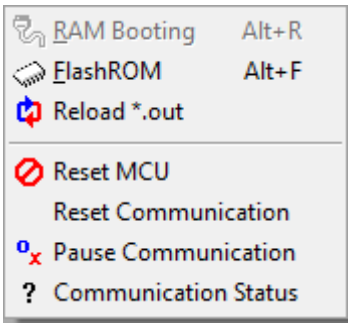
단계 4 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

단계 5 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 부트로더 프로그램이 지속 수행됨에 유의하세요.

주의) 지워지지 않은 섹터에 Program 할 때, 오동작의 가능성이 있습니다.

8.3.11 RA

MCU 메뉴 (Renesas RA)

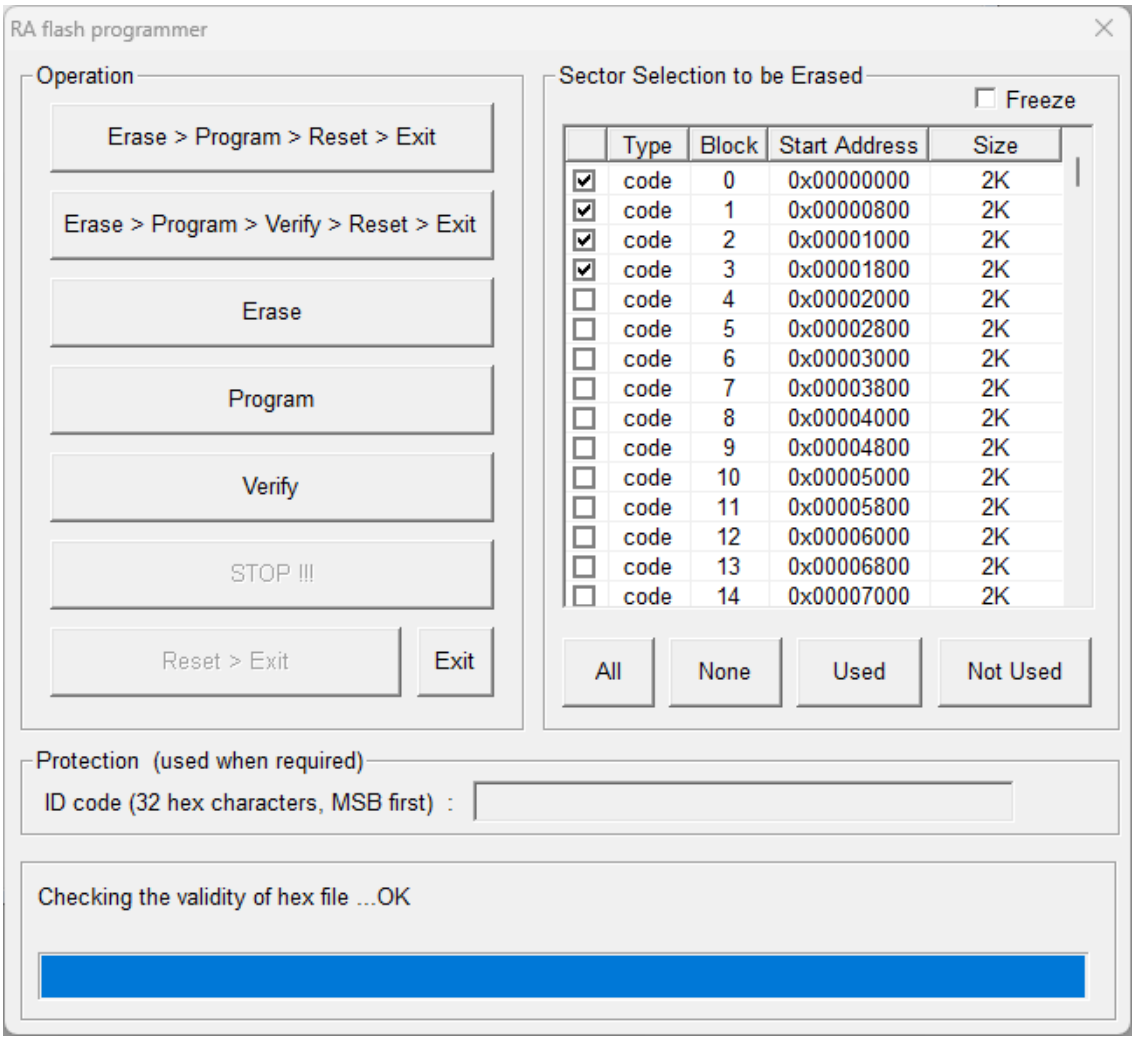


RAM Booting 메뉴

본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

사용자 프로그램을 플래시에 프로그래밍합니다. 단, RA0 시리즈는 예외로 플래시 프로그래밍이 지원되지 않습니다.
보호 기능이 적용되어 있는 영역은 프로그래밍이 불가할 수 있으니 주의 바랍니다.
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



사용 순서는 다음과 같습니다.

스텝 1 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용).

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

각 섹터의 체크 박스를 일일이 클릭하여 섹터별로 선택할 수도 있습니다.

Freeze 체크 박스를 선택하면 섹터 선택을 비활성화시킬 수 있습니다.

참고로 option flash 에 대해서는 erase 동작이 필요 없으므로 수행하지 않습니다.

스텝 2 : Erase, Program, Verify 버튼을 처음 사용시, MCU 리셋 이후 MCU 는 부트모드로 진입하게 합니다.

DLM(Device Lifecycle Management)을 사용하지 않는 MCU 가 locked 상태일 경우, ID code 를 입력해야 합니다.

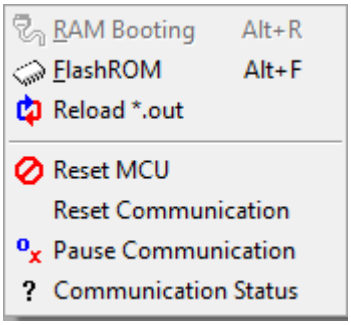
DLM 을 사용하는 MCU 에 대한 DLM 상태 변경은 지원되지 않습니다.

스텝 3 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

스텝 4 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 부트모드 프로그램이 지속 수행됨에 유의하세요.

8.3.12 RX

MCU 메뉴 (Renesas RX)



RAM Booting 메뉴

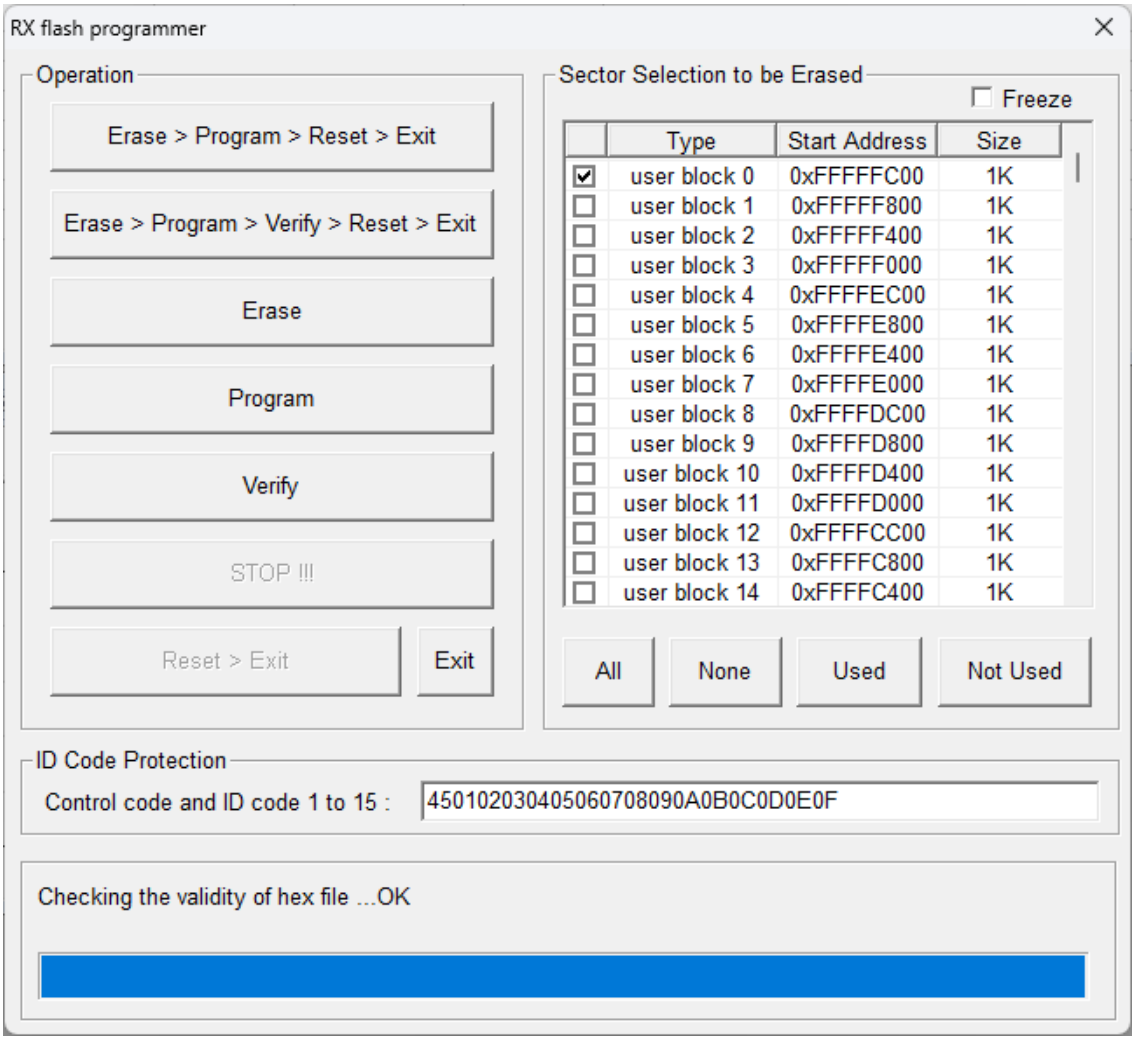
본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

사용자 프로그램을 플래시에 프로그래밍합니다.

단, 보호 기능이 적용된 영역이나 Trusted memory 영역은 지원되지 않으니 해당 기능이 해제된 상태에서 사용하시기 바랍니다.

easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



사용 순서는 다음과 같습니다.

스텝 1 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용).

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

각 섹터의 체크 박스를 일일이 클릭하여 섹터별로 선택할 수도 있습니다.

Freeze 체크 박스를 선택하면 섹터 선택을 비활성화시킬 수 있습니다.

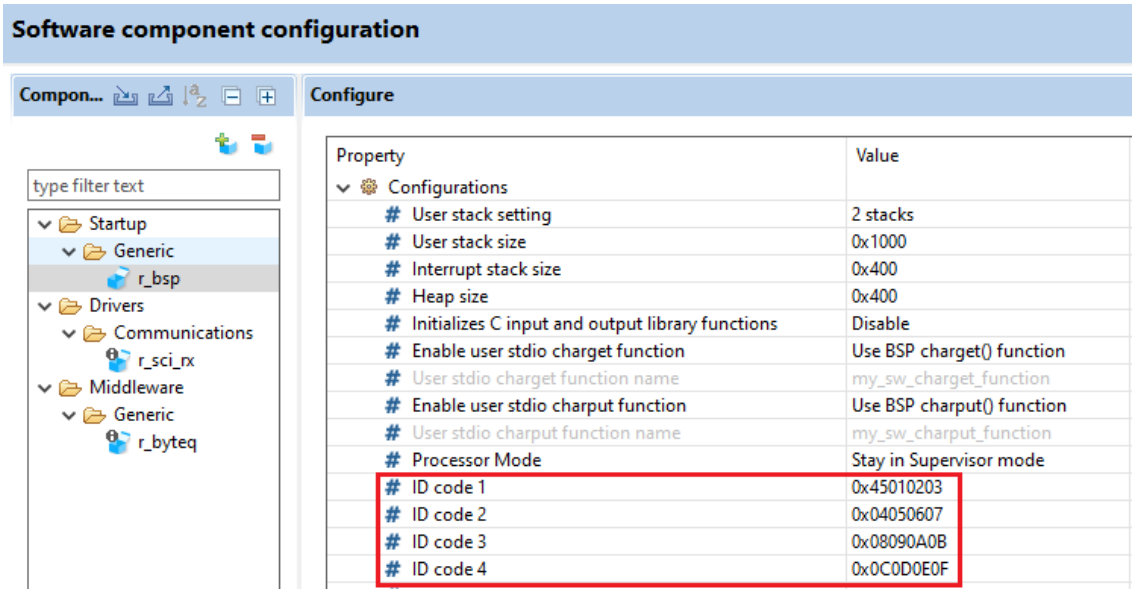
스텝 2 : Erase, Program, Verify 버튼을 처음 사용시, MCU 리셋 이후 MCU 는 부트모드로 진입하게 합니다.

스텝 3 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

스텝 4 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 부트모드 프로그램이 지속 수행됨에 유의하세요.

MCU 에서 Boot mode ID code protection 이 활성하되어 있을 경우, ID 코드를 입력해야 스텝 2 부트모드 진입이 가능합니다.

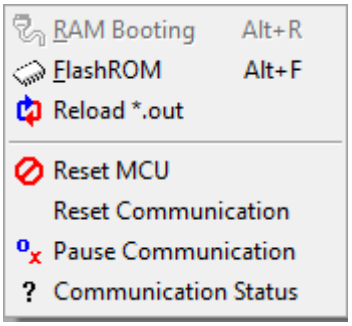
Smart Configurator 에서 아래 그림과 같이 ID 코드를 설정할 경우, 위의 대화상자 그림과 같이 코드 45010203040506070809 0A0B0C0D0E0F 를 입력하시기 바랍니다.



주의 사항 :

1. RX100, RX200 시리즈에서는 control ID 가 0x45 또는 0x52 이 아닌 경우, 부트 모드에 진입하기 전 전체 플래시가 지워지게 됨에 유의하세요.
2. RX64M, RX660, RX66T, RX71M, RX72T 시리즈의 경우 option setting memory 의 프로그래밍을 지원하지 않습니다.

8.3.13 TX, TXZ3



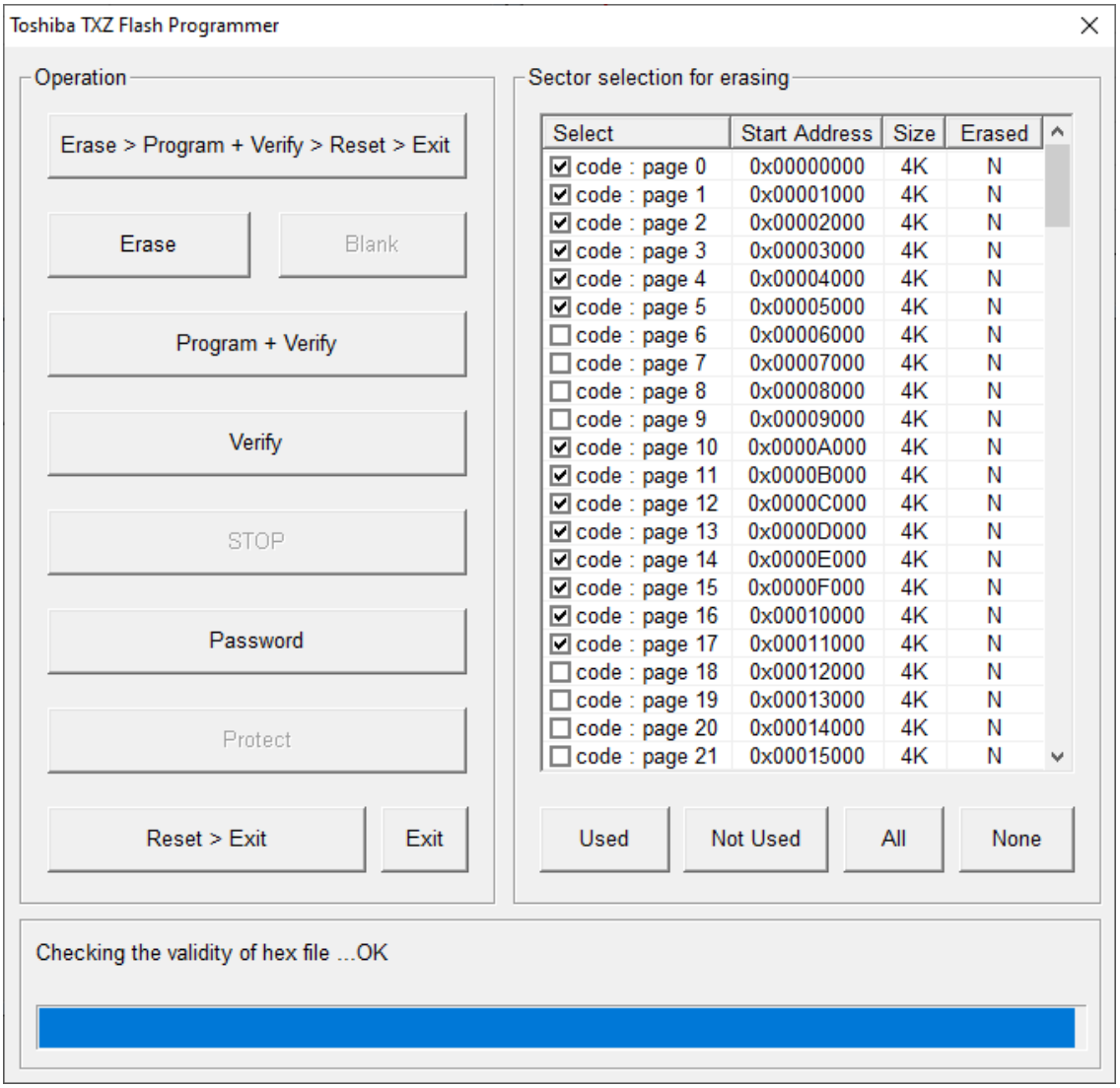
RAM Booting 메뉴

본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

사용자 프로그램을 플래시에 프로그래밍합니다.

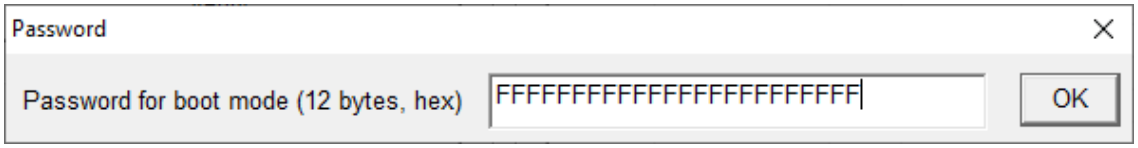
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.



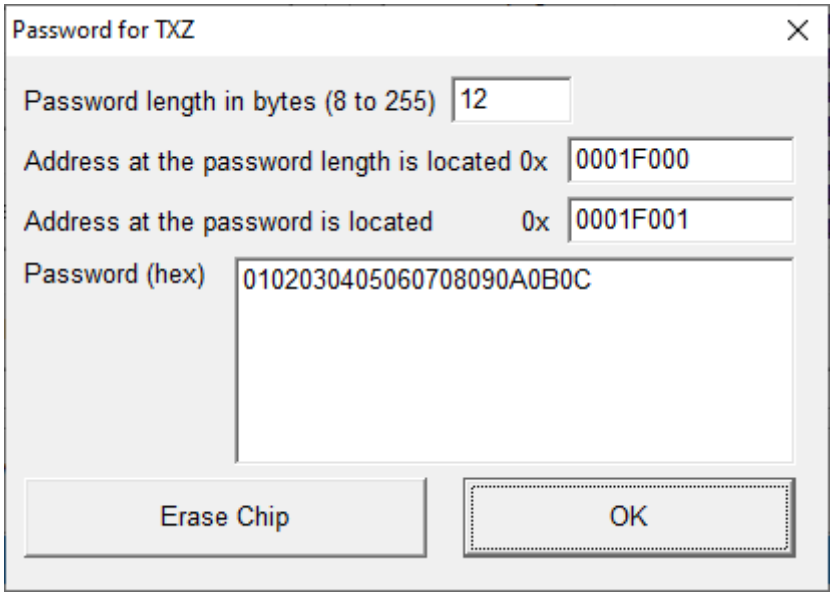
사용 순서는 다음과 같습니다.

단계 1 : Password 버튼을 클릭하여 MCU 가 싱글 부트 모드 진입하기 위한 비밀번호를 설정합니다.

TX 시리즈 경우 하기 대화상자와 같이 12 바이트 값을 입력하며 초기값은 FFFFFFFFFFFFFFFFFFFFFFFF 입니다.



TXZ3 시리즈의 경우 하기 대화상자에 관련 항목을 입력해야 합니다.



단계 2 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용). 체크 박스를 사용하여 섹터별로 선택할 수도 있습니다.

Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.

단계 3 : Erase, Program+Verify, Verify 버튼을 처음 사용시 MCU 리셋이 걸리고 MCU 는 부트로더로 진입하게 합니다.

단계 4 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

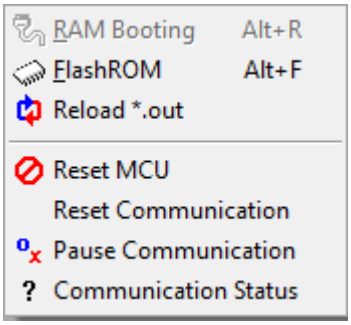
단계 5 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 부트로더 프로그램이 지속 수행됨에 유의하세요.

주의) 지워지지 않은 섹터에 Program 할 때, 오동작의 가능성이 있습니다.

주의) Blank, Protect 버튼은 동작하지 않습니다.

8.3.14 LPC

MCU 메뉴 (NXP LPC1x00)

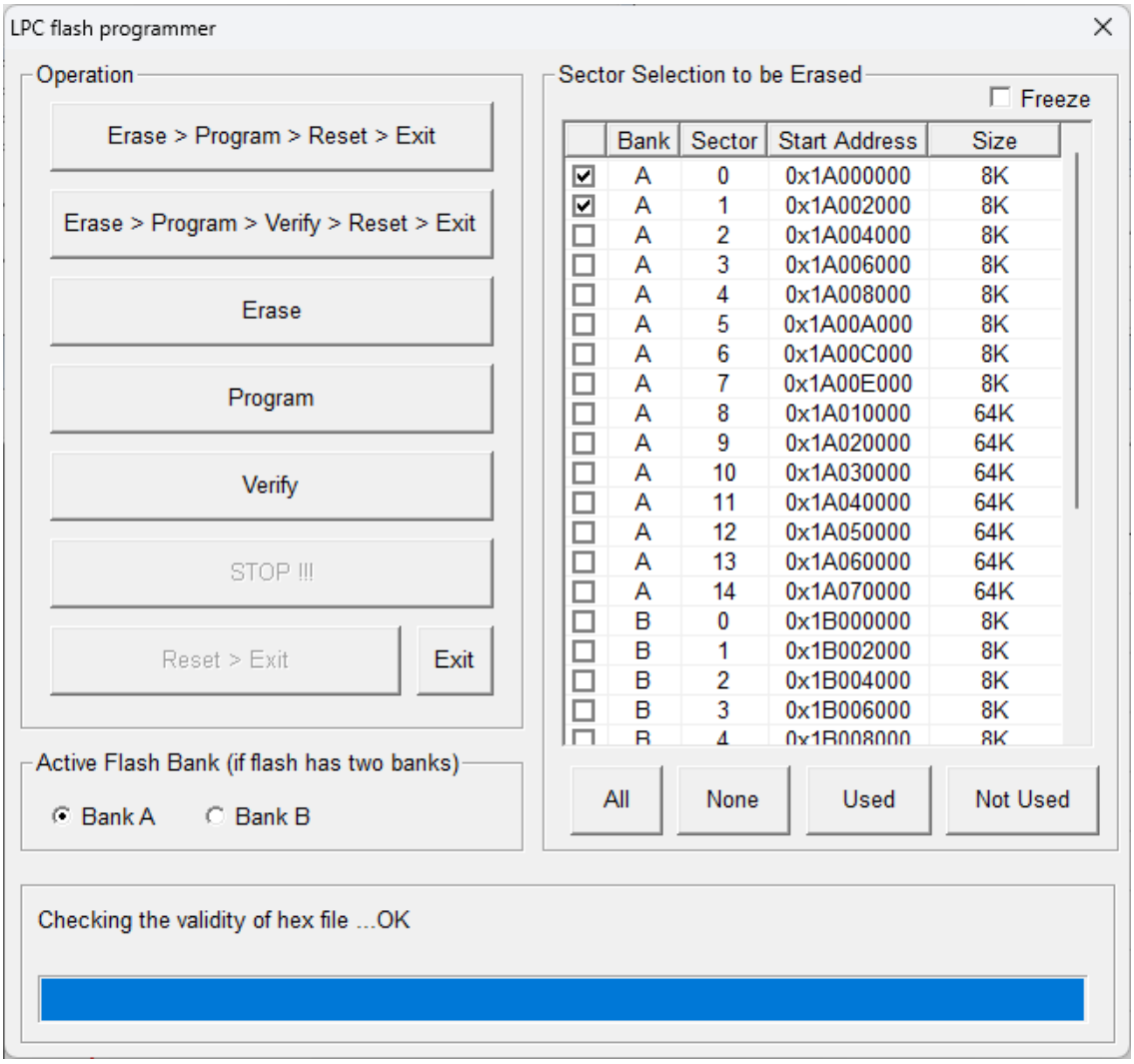


RAM Booting 메뉴

본 메뉴는 지원되지 않습니다.

Flash ROM 메뉴

사용자 프로그램을 플래시에 프로그래밍합니다.
단, 보호 기능이 해제된 상태에서 사용하시기 바랍니다.
easyDSP 의 모니터링 기능은 일시 정지되며, 다음과 같은 대화상자가 나타납니다.

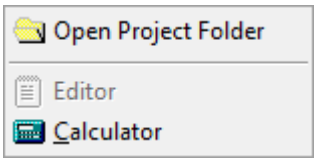


사용 순서는 다음과 같습니다.

스텝 1 : 플래시 뱅크가 2 개일 경우 (일부 LPC1800), 사용자 프로그램이 수행될 뱅크를 선정합니다.
스텝 2 : Erase 대상이 되는 플래시 섹터를 선정합니다 (All, None, Used, Not Used 버튼 활용).
Used 버튼은 사용자 프로그램이 사용하는 모든 섹터를 선택합니다. Not Used 버튼은 그 반대입니다.
각 섹터의 체크 박스를 일일이 클릭하여 섹터별로 선택할 수도 있습니다.
Freeze 체크 박스를 선택하면 섹터 선택을 비활성화시킬 수 있습니다.
스텝 3 : Erase, Program, Verify 버튼을 처음 사용시, MCU 는 ISP 모드로 진입 (MCU 리셋 수반) 하게 합니다.
스텝 4 : 각종 버튼을 사용하여 플래시 동작을 수행합니다.

스텝 5 : 'Reset>Exit' 버튼으로 대화상자를 나가면서 사용자 프로그램을 수행합니다. 리셋 없이 대화상자를 나가면 부트모드 프로그램이 지속 수행됨에 유의하세요.

8.4 Tools



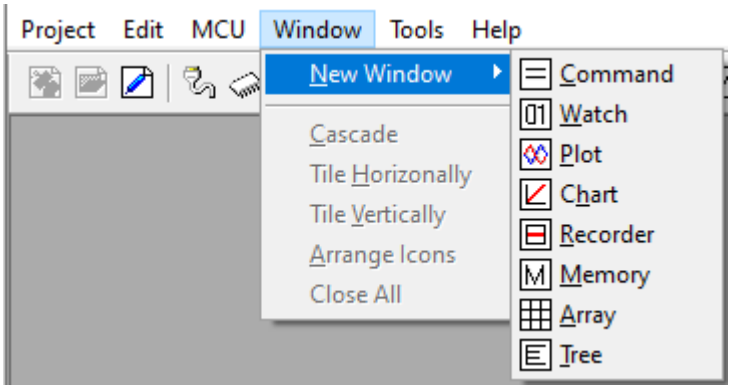
몇 가지 도구들이 있습니다.

'Open Project Directory' : 현재 프로젝트 폴더를 엽니다.

'Editor': 프로젝트에서 설정된 에디터를 엽니다.

'Calculator' : 윈도우즈 계산기를 실행합니다.

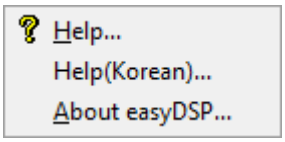
8.5 Window



여러 윈도우들을 열고/닫고/배열하고/선택합니다.

8.6 Help

Help 메뉴

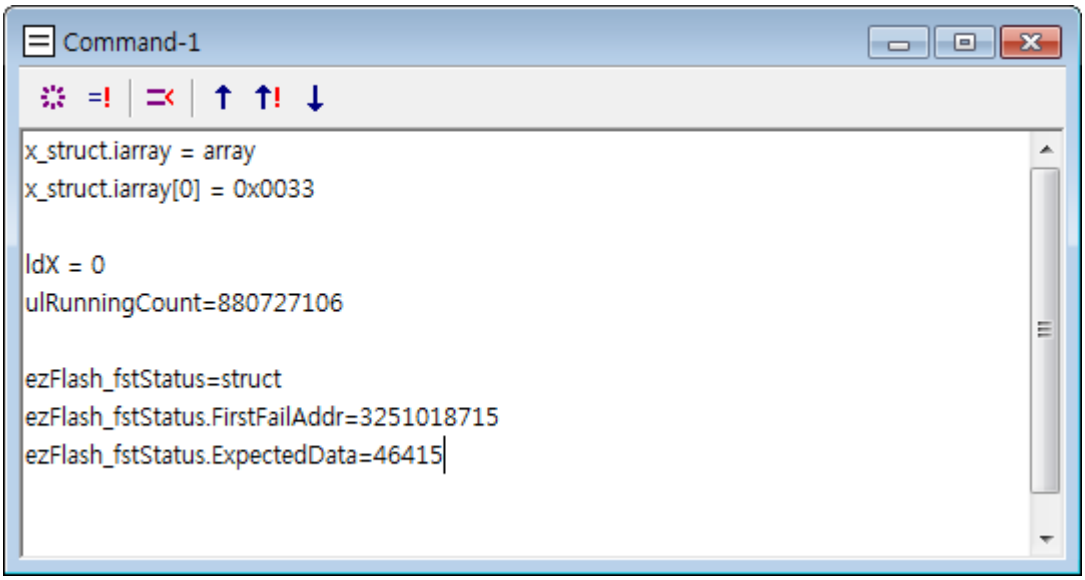


'Help...' : 본 도움말 파일을 엽니다

'About easyDSP...' : 기본 정보를 표시합니다

9. 윈도우

9.1 Command



커맨드 윈도우는 MCU 의 각종 변수의 값을 읽거나 쓰는 용도에 사용됩니다. 커맨드윈도우 내에서 'help'라고 타이핑하고 엔터를 치면 명령의 각종 형식에 대해 표시됩니다. 각종 명령의 수행은 사용자가 해당 라인에서 엔터키를 누름으로써 수행됩니다.

도구바( )

현재 커서가 위치한 라인을 업데이트합니다. 또는 구간이 선택되었을 때는 선택된 구간을 업데이트합니다. 마우스 오른쪽 버튼을 클릭해도 동일한 기능이 수행됩니다.

현재 커서가 위치한 라인의 명령을 수행합니다. 또는 구간이 선택되어 있을 시 해당 구간을 수행합니다. 만약 전 구간을 수행하고자 할 경우에는, CTRL-A 를 입력하여 전 구간을 선택한 후 버튼을 누르면 됩니다.

커맨드 윈도우의 현재 커서의 위치에 새로운 라인을 삽입합니다. 이 기능은 CTRL-Enter 키의 입력과 동일합니다.

대화상자로부터 선택한 커맨드파일(*.cmd)의 내용을 커맨드 윈도우로 복사합니다. 수행되는 내용은 없습니다. 커맨드파일이란 커맨드 윈도우가 지원하는 명령을 모아둔 파일입니다. 사용자가 자주 사용하는 명령을 커맨드 파일로 작성한 후 용이하게 사용할 수 있습니다. 이 명령어는 'r 파일'의 형식으로 타이핑하므로서도 지원됩니다.

대화상자로부터 선택한 커맨드파일(*.cmd)을 수행합니다. 또한 이 명령어는 커맨드 윈도우에서 'I 파일'의 형식으로도 지원됩니다.

커맨드 윈도우의 내용을 파일로 저장합니다. 블럭이 선택되어 있을 시 블럭만을 저장할 수 있습니다. 이 기능은 커맨드파일을 생성할 때 유용합니다. 또는 현재 실험 상태를 저장할 때 유용합니다.

지원되는 명령어

진수 표시	
dec var	var 변수를 십진수로 표시합니다 (기본성질)
hex var	var 변수를 16 진수로 표시합니다
bin var	var 변수를 2 진수로 표시합니다

대입 및 표시	
var =	var 변수의 값을 표시합니다
&var =	var 변수의 번지값을 표시합니다
*var =	var 변수가 기본형 변수의 포인터일 경우 포인터가 가리키는 변수의 값을 표시합니다. 단 Arm 계열 MCU 에서는 지원되지 않습니다.
(*var).x =	var 변수가 구조체/공용체 변수의 포인터일 경우 포인터가 가리키는 변수 x 의 값을 표시합니다. 단 Arm 계열 MCU 에서는 지원되지 않습니다.
var = 숫자	var 변수에 주어진 값을 대입합니다 16 진수 숫자는 0x*** 형식으로 입력 가능합니다 2 진수 형식의 숫자는 지원되지 않습니다. 또한 var 변수가 실수일 경우, 다음과 같은 형식도 지원됩니다 3e-3, 23K, 23m, 0.34p
var1 = &var2	var1 변수의 값을 var2 변수의 번지로 대입합니다 이때 var1 변수는 정수형이어야 합니다
*var = 숫자 또는 <수식>	var 변수가 기본형 변수의 포인터일 경우 포인터가 가리키는 변수에 값 또는 수식을 대입합니다. 단 Arm 계열 MCU 에서는 지원되지 않습니다.
(*var).x = 숫자 또는 <수식>	var 변수가 구조체/공용체 변수의 포인터일 경우 포인터가 가리키는 변수 x 에 값 또는 수식을 대입합니다. 단 Arm 계열 MCU 에서는 지원되지 않습니다.
var = '문자'	var 변수가 char 또는 unsigned char 형식일 경우, 'A'와 같이 문자형식으로 입력 가능합니다. 문자형식으로 변수값을 표시하기 위해서는 프로젝트 설정에서 miscellaneous 탭의 'Display printable ...' 옵션을 활성화하시기 바랍니다.
var = <수식>	var 변수의 값을 <>안의 수식의 연산결과값으로 대입합니다 수식의 예는 다음과 같습니다

	<e^pi-pi^e>, <1/ln(x)/x>, <exp(-1/pow(x/100,2))>
var = 숫자 Qn	var 가 16 비트 정수일 경우에는 n=1-15, 32 비트 정수일 때에는 n=1-31. Q 포맷에 따라 처리됨
var = 숫자 Q	var 변수에 Q 포맷이 미리 설정되어 있는 경우, 해당 Q 포맷에 따라 처리됨. 즉, var 변수가 Q12 일 경우, var=3.14Q 는 var=3.14Q12 와 같음
var = <수식>Qn var = <수식>Q	<>안의 수식이 자동계산되어 숫자 Qn, 숫자 Q 형식과 마찬가지로 처리됨

기타	
clear	커맨드 윈도우의 모든 내용을 삭제합니다
//	한 라인 중 '//' 이후의 문장은 주석으로 처리합니다
help	커맨드 윈도우 내에서 사용되는 명령어를 설명합니다
l 커맨드파일	커맨드 파일을 수행(load)합니다. 즉, 커맨드파일의 내용을 line to line 으로 실행합니다.
r 커맨드파일	커맨드파일의 내용을 커맨드윈도우에 복사할 뿐, 수행되는 것은 없습니다.
skip	커맨드 파일을 수행할 시, 커맨드 파일내의 skip 명령 뒤의 커맨드는 수행하지 않습니다.

주의사항:

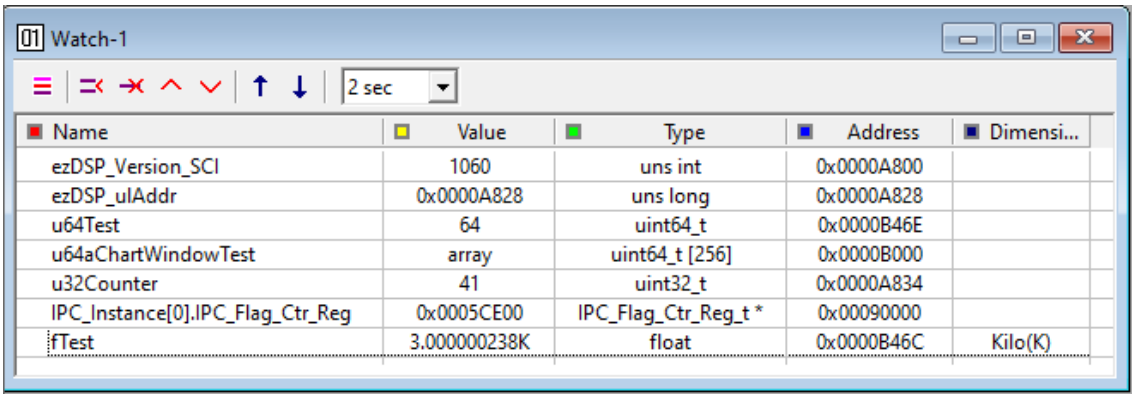
한 라인에서 수행될 수 있는 커맨드의 길이는 300 을 넘지 말아야합니다.

모든 명령어는 소문자입니다.

디멘젼 기능 및 Q 기능

오직 Watch 윈도우에서만 실수형 변수에 디멘젼(dimmension) 기능을, 정수형 변수에 Q 기능을 설정할 수 있습니다. 해당 변수를 리드/라이트할 시에는 동일한 원칙이 본 Command 윈도우에서도 적용됩니다. 자세한 사항은 Watch 윈도우를 참조하세요.

9.2 Watch



Watch 윈도우 내에서 MCU 프로그램내의 변수를 모니터링 하거나 변경할 수 있습니다.
MCU 의 통신 부하를 최소화하기 위해 보이는 항목에 대한 값만 갱신됨에 유의하세요.
버튼의 기능은 다음과 같습니다.

: 토글되는 버튼으로 모든 변수 또는 사용자가 등록한 변수만을 표시합니다

: 변수 등록 (Insert 키와 동일 작동)

: 변수 삭제 (Delete 키와 동일 작동)

: 변수 위치 한단계 위로

: 변수 위치 한단계 아래로

: 기저장된 변수이름 리스트를 로딩합니다. 변수 값을 변경하는 것은 아닙니다.

: 현재 창의 변수 이름 및 변수 값을 저장합니다. cmd 확장자로 저장될 경우, 차후 Command 윈도우에서 로딩하여 현재 Watch 윈도우상의 변수 값을 다시 변수에 대입 가능합니다.

제어 보드의 환경 설정에 관련된 일련의 변수들을 처리할 때 용이하게 사용 가능합니다.

각 항에 대한 설명은 다음과 같습니다.

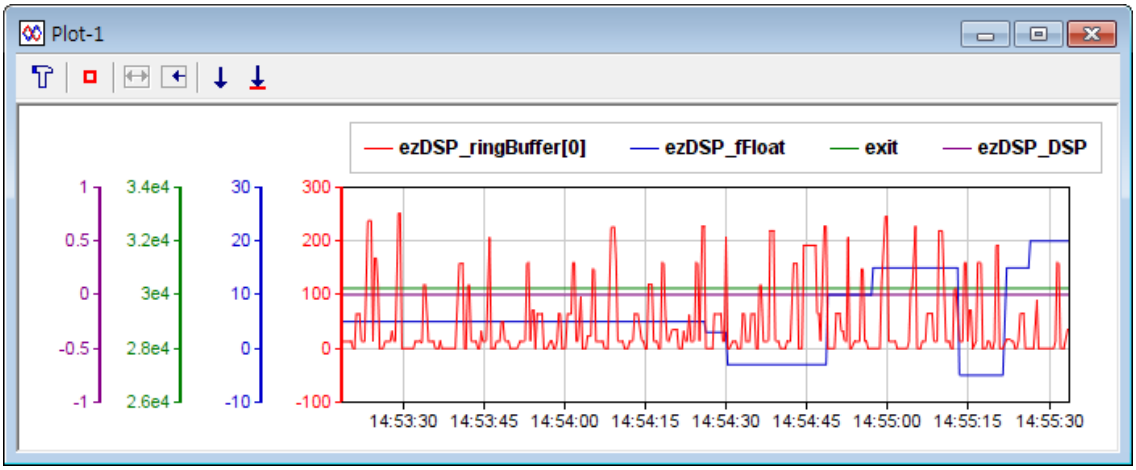
항목	기능
Name	변수 이름을 표시합니다. 또한 TI C28x MCU 에서 포인터 변수의 역참조 연산자를 사용할 수 있습니다. 예를 들어, *포인터 변수 : 포인터 변수가 기본형 변수를 가리킬 때 (*포인터 변수) : 포인터 변수가 구조체/공용체 변수를 가리킬 때
Value	변수 값을 표시합니다. 사용자가 마우스 왼쪽 버튼을 클릭함으로써 또는 엔터키를 누름으로써 해당 변수의 값을 변경시킬 수 있습니다. 변수 변경시 <수식>형식을 사용하실 수 있습니다. 사용자가 마우스 오른쪽 버튼을 클릭함으로써 진수 모드를 토글시킬 수 있습니다 (10 진수 => 16 진수

	=> 2 진수 = > 10 진수....). 16 진수 표시모드일 경우 0x**로 표시됩니다. 2 진수 표시의 경우 0b**로 표시됩니다.단, 포인터 변수의 경우에는 16 진수로 표시가 고정됩니다. 또한, 사용자가 특별한 디멘젼을 등록한 경우, 100u, 1K...등으로 표시됩니다. 변수값 입력시 지원되는 형식에 대해서는 Command 윈도우의 도움말을 참조하시기 바랍니다.
Type	변수의 형태가 표시됩니다.
Address	변수의 번지가 표시됩니다.
Dimension	변수의 형식에 따라 디멘젼 또는 Q 형식을 지정할 수 있습니다. **디멘젼** 실수형 변수 경우, 디멘젼이 표시됩니다. 사용자가 마우스 왼쪽 버튼을 클릭함으로써 디멘젼의 값을 변경할 수 있습니다. 디멘젼은 변수 읽기 뿐 아니라 변수값 쓰기 시에도 사용할 수 있습니다. 디멘젼 p = pico (10^{-12}) 디멘젼 n = nano (10^{-9}) 디멘젼 u = micro (10^{-6}) 디멘젼 m = mili (10^{-3}) 디멘젼 K = Kilo (10^{3}) 디멘젼 M = Mega (10^{6}) 디멘젼 G = Giga (10^{9}) **Q 형식** 정수형 변수에는 Q 기능이 제공됩니다. 특히 고정소수점 MCU 에 유용한 기능입니다. 16 비트 정수형 변수에는 Q0 부터 Q15 까지를, 32 비트 정수형 변수에는 Q0 부터 Q30 까지를 설정할 수 있습니다. Q 설정된 변수의 리드/라이트시 아래 문법을 사용해야 합니다. **즉, Qn 형식이 지정된 정수형은 마치 실수형처럼 읽고 쓸 수 있습니다. Plot, Chart 윈도우에서도 Qn 형식이 지정된 정수는 마치 실수처럼 표시합니다.** - 변수 리드시 Q 변환된 변수의 실수값 뒤에 Qn 을 붙여서 표시합니다. 예를 들어 정수형 변수 var1 이 Q15 로 설정되어 있을 경우, '3.14Q15'로 표기됩니다. 이는 해당 변수가 Q15 로 설정되어 있다는 것을 환기시키기 위한 것입니다. 디폴트인 Q0 에 대해서는 변수값 뒤에 Q0 을 붙이지 않으며, 당연히 정수형으로 변수값을 표시합니다. - 변수 라이트시 Q0(디폴트)로 설정된 변수(정수형)는 , 하기의 2 가지 형식의 값으로 라이트 할 수 있다.

	쓰기 형식 1: var1 = 314, 당연히 314 로 라이트 됨. 쓰기 형식 2: var1 = 3.14Qn : 실수형 3.14 를 Qn 형으로 변환해서 라이트됨 Qn(n=1...31)으로 설정된 변수는 하기 2 가지 형식으로 라이트 할 수 있다 쓰기 형식 1: var1 = 3.14Qn : 실수형 3.14 를 Qn 형으로 변환해서 라이트됨. 주의해야 할 것은 변수자체는 Q15 로 설정되어 있다 하더라도 var1 = 3.14Q14 로 라이트하면, 3.14Q14 로 라이트된다. 쓰기 형식 2: var1 = 3.14Q : Q 뒤에 n 이 없으면 설정된 Qn 값으로 라이트됨. 쓰기 형식 3: var1 = <cos(pi/3)>Q31 : <>안의 수식이 0.5 이므로 0.5Q31 과 동일하게 처리됨.

9.3 Plot

Plot 윈도우



Plot 창에서는 등록된 변수 값을 실시간 그래프로 표시하며, 데이타를 일정 시간 동안 저장합니다. 샘플링 시간 대비 느리게 변화하는 변수에 대해 리코더 역할을 수행합니다.

Qn 형식이 지정된 정수형은 마치 실수형처럼 표시합니다. 예를 들어, Q31 이 지정된 32 비트 정수형 변수는 +1/-1 사이의 실수형 값으로 표시됩니다.

툴바 설명

: 표시될 변수 및 기타 관련 정보를 설정합니다. 다음과 같은 대화상자에서 변수이름, Y 축 표시 범위의 최소/최대/자동값 또는 표시 모드 등을 설정할 수 있습니다.

샘플링 주기 Sampling interval 의 최소값은 5msec 입니다. 이 주기로 등록된 변수를 읽어서 최대 Total plot period 동안 그래프로 표시합니다.

설정된 환경은 'Save'/'Load'버튼을 통해 저장/로딩할 수 있습니다.

색깔 및 심볼은 채널별로 기정의 되어 있습니다.

채널 1 번 : 빨강 - 동그라미

채널 2 번 : 파랑 - 네모

채널 3 번 : 초록 - 삼각

채널 4 번 : 보라 - 다이아몬드

채널 5 번 : 검정 - 오른방향 삼각

채널 6 번 : 연두 - 왼방향 삼각

채널 7 번 : 회색 - 십자가

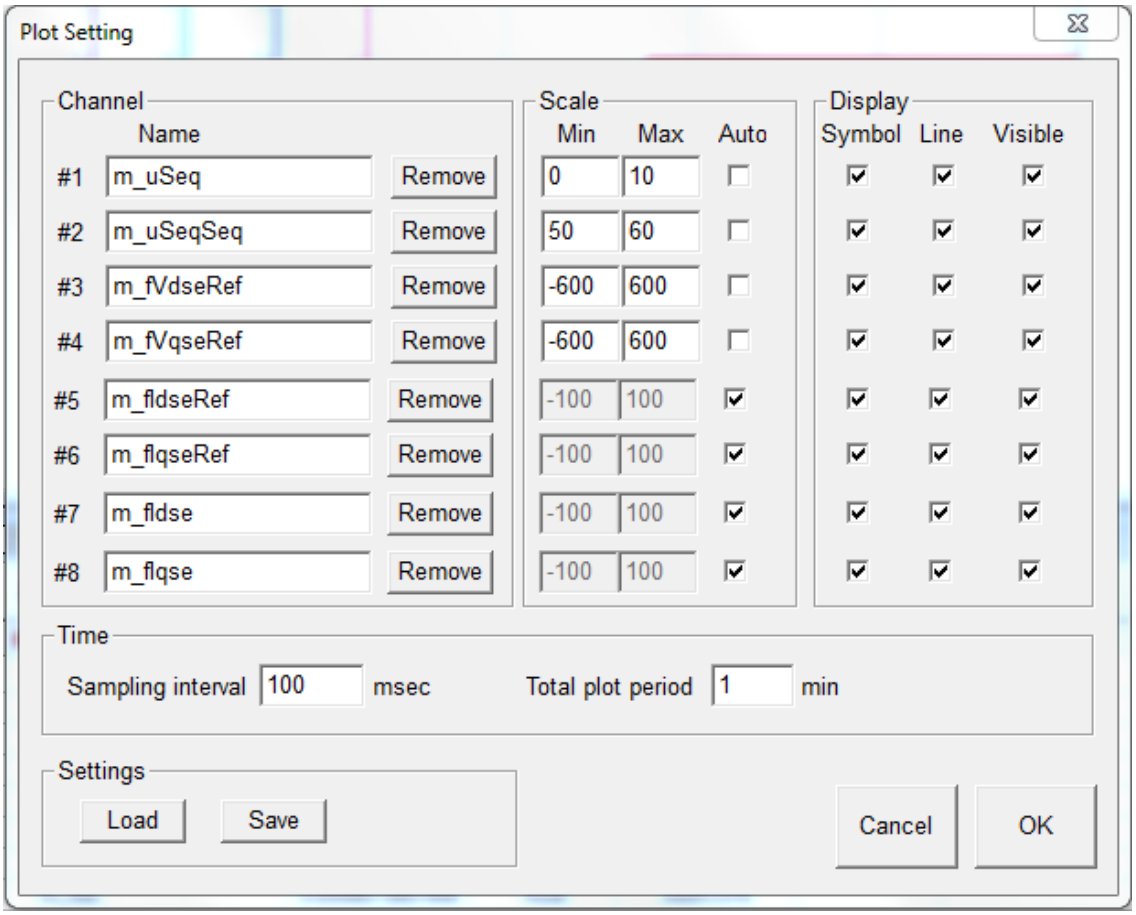
채널 8 번 : 오렌지 - X 글자

주의 사항) 하기 원인에 의하여 대부분의 경우 설정된 주기보다 긴 주기에 변수값을 읽습니다.

1. 윈도우즈의 시스템 타이머 최소 정밀도는 약 10msec 로, 실제적인 샘플링 시간은 설정된 값과 다를 수 있습니다.

2. 데이타의 개수가 많을 수록 그래픽 처리 등등에 시간이 더욱 소요되여 설정된 주기보다 훨씬 길어질 수 있습니다.

3. 사용자 PC 의 메모리가 충분하다면, 변수당 최대 4,294,967,295 개를 표시/저장할 수 있습니다.



 (토글버튼) : 그래프를 잠시 멈춥니다. / 멈춰진 그래프를 다시 진행시킵니다.

 : 줌(Zoom) 사용 중에 전체 데이타를 표시할 경우 사용합니다.

전체 데이타란, 사용자가 지정한 'Total plot period'를 지원하기 위해 할당된 전체 메모리를 의미합니다.

 : 줌(Zoom) 사용 중에 최신 데이타를 표시할 경우 사용합니다.

최신 데이타란, 현재 plot 창의 사이즈 해상도 숫자만큼을 의미합니다.

 : 화면을 그래픽 파일(bmp, jpg, png 형식 지원)로 저장하거나, 데이타를 텍스트 파일(csv 형식)로 저장합니다.

하기 예제에서는 m_uSeq 변수의 값을 저장한 것입니다.

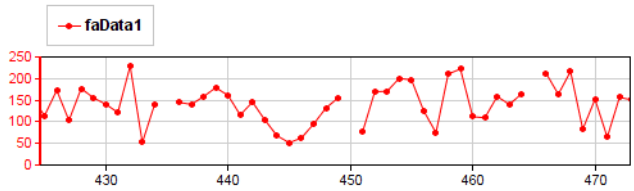
맨 왼쪽 3 칸은 절대 시간이며, 다음 칸은 첫 저장 데이타시간 기준 경과 시간이며, 마지막은 값이 저장된 것입니다.

m_uSeq				
date(year-month-day)	time(hour-min-sec)	time(mili_sec)	elapsed time(mili_sec)	value
2017-09-04	12:48:42	223	0	2
2017-09-04	12:48:42	359	136	2
2017-09-04	12:48:42	475	252	2
2017-09-04	12:48:42	597	374	2
2017-09-04	12:48:42	722	499	2
2017-09-04	12:48:42	848	625	2
2017-09-04	12:48:42	976	753	2
2017-09-04	12:48:43	98	875	2
2017-09-04	12:48:43	226	1003	2
2017-09-04	12:48:43	346	1123	2
2017-09-04	12:48:43	470	1247	2
2017-09-04	12:48:43	584	1361	2
2017-09-04	12:48:43	706	1483	2

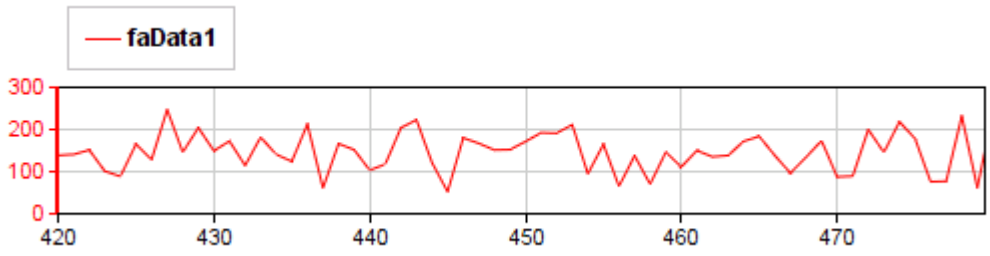
 : 화면상의 데이타를 레코드 파일(*.rec 파일)로 저장합니다. 추후 레코드윈도우에서 읽어 들일 수 있습니다.

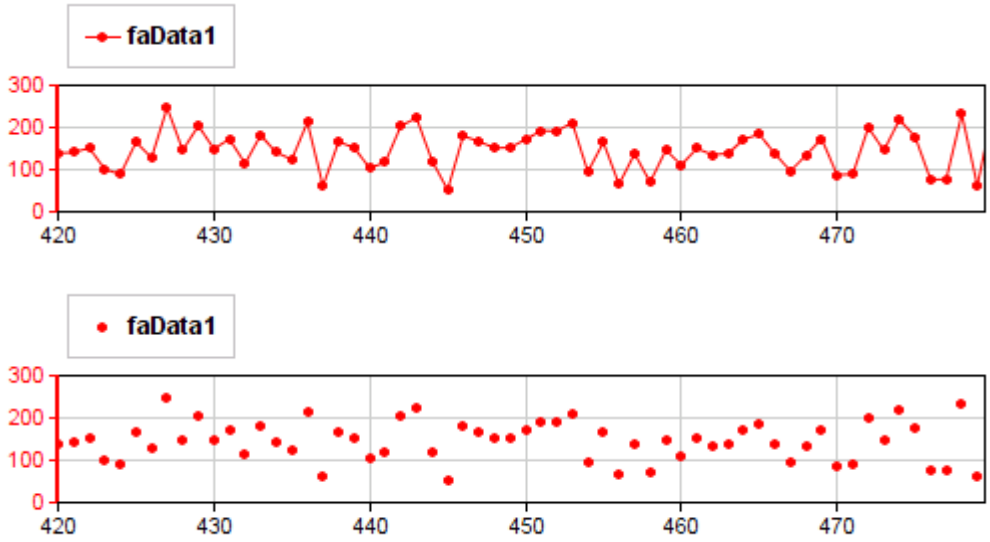
유용한 기능들

- 마우스 커서를 그래프에 위치시키면 해당 지점의 데이타 값을 표시하여 줍니다 (Tooltip 기능).
- 통신 오류가 발생된 지점은 공백처리 합니다. 아래처럼 선이 끊기게 표시됨. 동일하게 일정 기간 동안 사용자의 고의로 통신을 중지하였을 경우(예:그래프를 멈춤)에도 공백처리 됩니다.

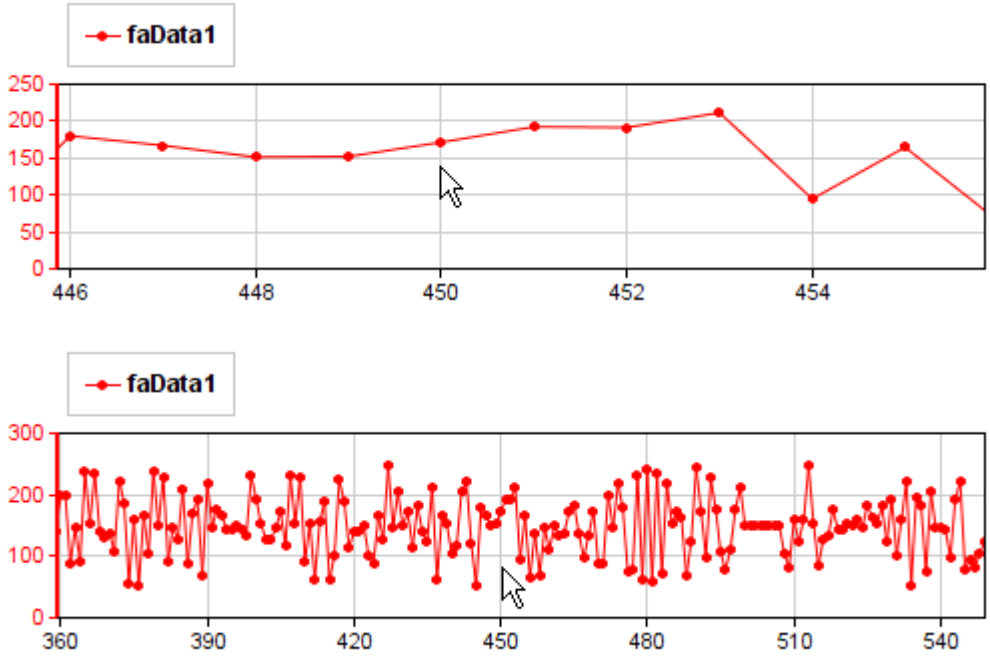


- 심볼 및 선을 표시/미표시 선택함으로써 다양한 표시가 가능합니다.

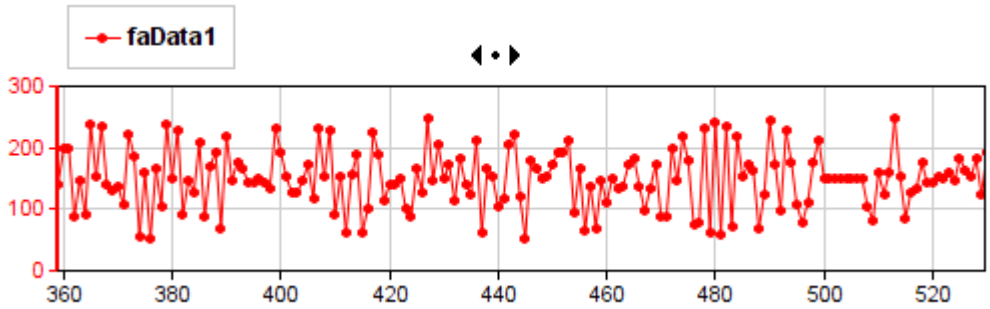




- 마우스 휠을 조작하여 X 축을 확대/축소시킬 수 있습니다.



- Zoom 상태에서 마우스 왼쪽 버튼을 누르고 화면을 drag 시킬 수 있습니다. 이때 마우스 커서 모양이 변경됩니다.



9.4 Chart

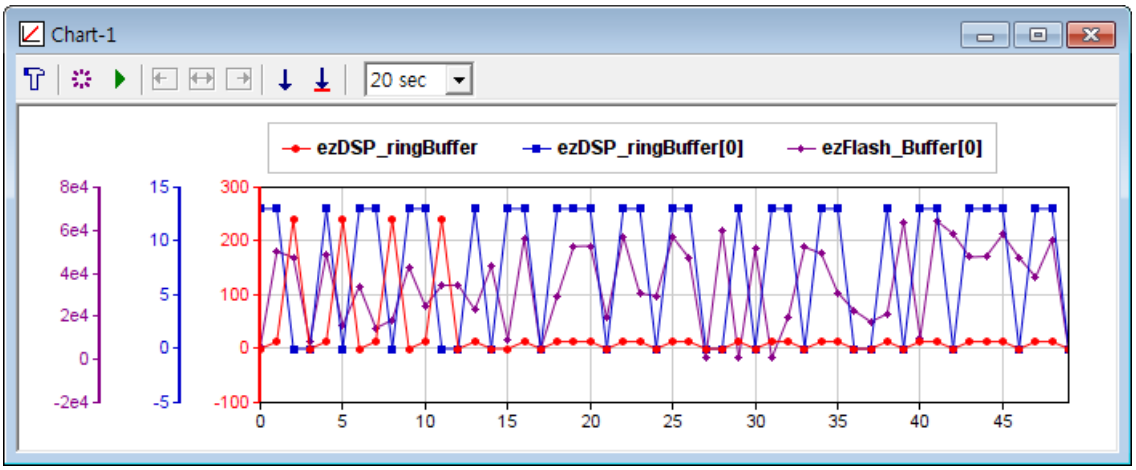


Chart 윈도우는 사용자가 MCU 내 메모리에 일정 공간을 마련해 놓고 (일차원 배열 형식으로), 이 공간에 필요한 데이타를 저장한 후, 나중에 이를 정밀 관찰하기에 유용합니다. 사용자 시스템의 메모리가 충분하다면, 동작 중에 관찰이 필요한 변수를 메모리에 연속적으로 저장해 놓은 후, 이를 나중에 볼 수 있습니다. 이로서 일정 부분 오실로스코프 역할을 대신할 수 있습니다.

Chart 윈도우에서는 해당 데이타에 대한 쓰기 기능은 제공하지 않습니다.

Qn 형식이 지정된 정수형은 마치 실수형처럼 표시합니다. (즉, Q31 이 지정된 32 비트 정수형 변수는 +1/-1 사이의 실수값으로 표시됩니다)

툴바 설명

: 버튼 클릭시 다음과 같은 대화상자가 표시되면서, 최대 8 개까지의 변수를 등록합니다.

'Channel' : 일차원 배열 변수를 선택합니다. 일차원 배열 변수(ex, one_dim_var)을 선택할 경우, 해당 변수의 원소 개수가 자동 선택되어집니다.

'Scale' : 변수값이 표기될 Y 축 범위를 설정합니다. Auto 를 설정하면 자동 표시됩니다.

'Display' : 심볼/선을 표시할 지를 선택합니다. Visible 항목을 비활성화시, 해당 변수가 표시되지 않습니다 (단, 데이타 취합은 지속됩니다).

'Enable fast reading' : MCU 리소스 여유가 있어 빠른 통신이 가능할 때, 본 옵션을 활성화하면 빠른 통신을 통해 그래프를 신속히 갱신합니다.

실패시 그래프에 아무 것도 표시되지 않습니다.

빠른 통신을 위한 리소스가 충분치 않은 경우 또는 기타 사유로 동작이 되지 않을 경우 비활성화하여 주세요.

색깔 및 심볼은 채널별로 기정의 되어 있습니다.

채널 1 번 : 빨강 - 동그라미

채널 2 번 : 파랑 - 네모

채널 3 번 : 초록 - 삼각

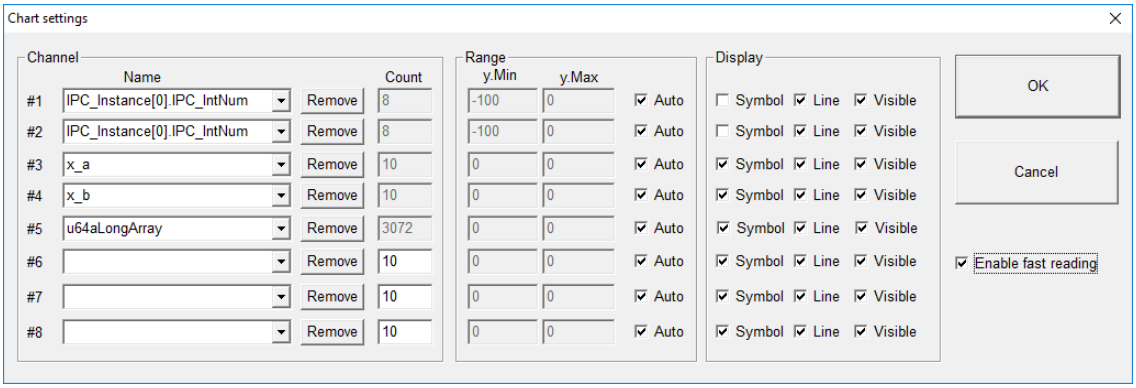
채널 4 번 : 보라 - 다이아몬드

채널 5 번 : 검정 - 오른방향 삼각

채널 6 번 : 연두 - 왼방향 삼각

채널 7 번 : 회색 - 십자가

채널 8 번 : 오렌지 - X 글자



: 그래프를 일회성으로 업데이트 합니다. 사용하는 데이타 개수가 매우 큰 경우에는 이를 샘플링 간격마다 표시하는 것이 매우 부담스럽습니다. 이 경우에는 필요시에만 업데이트해서 보는 것이 유용합니다.

(토글버튼) : 그래프를 잠시 멈춥니다. / 멈춰진 그래프를 다시 진행시킵니다.

: 줌(Zoom) 사용 중에 화면 왼쪽 데이타를 표시합니다.

: 줌(Zoom) 사용 중에 전체 데이타를 표시합니다.

: 줌(Zoom) 사용 중에 화면 오른쪽 데이타를 표시합니다.

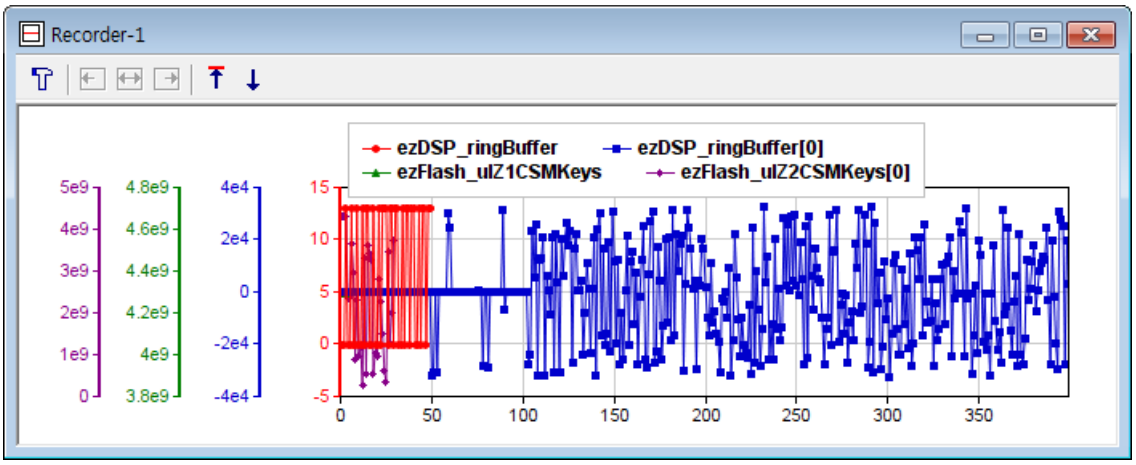
: 화면을 그래픽 파일(bmp, jpg, png 형식 지원)로 저장하거나, 데이타를 텍스트 파일(csv 형식, 추후 엑셀로 처리 편리)로 저장합니다.

: 화면상의 데이타를 레코드 파일(*.rec 파일)로 저장합니다. 추후 레코드 윈도우에서 읽어 들일 수 있습니다.

유용한 기능들

- Plot 창과 동일하게 그래프를 유용하게 사용하는 방법에 대해서는 링크 를 확인하세요.

9.5 Record



Record 윈도우는 사용자가 Chart 또는 Plot 윈도우를 통해 저장한 레코드 파일(확장자 rec)을 읽어서 그 내용을 그래프로 표시합니다. 중요한 데이타를 Chart 또는 Plot 윈도우를 통해 레코드 파일로 저장한 후 Record 윈도우로 관찰하면서 유용하게 사용되길 희망합니다.
본 창이 열릴 때 자동적으로 이전 사용자 설정 내용(레코드 파일, 확대 영역, 선 관련 각종 설정)을 복원합니다.

툴바 설명

 : 다음과 같은 대화상자가 표시되며, 먼저 레코드 파일 이름 및 저장 시기를 표시합니다. 기타 설정 부분은 Chart 또는 Plot 윈도우와 동일합니다.

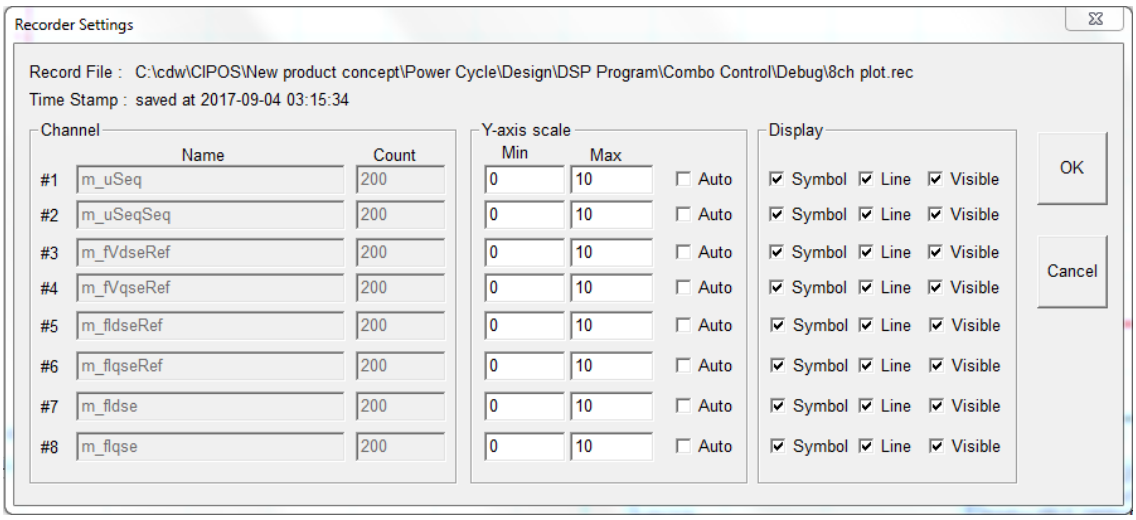
'Channel' : 변수 이름 및 저장 데이타 개수이며, 레코드 파일의 내용을 그대로 가져온 것으로, 사용자가 변경할 수 없습니다.

'Scale' : 변수값이 표기될 Y 축 범위를 설정합니다. Auto 를 설정하면 자동 표시됩니다.

'Display' : 심볼/선을 표시할 지를 선택합니다. Visible 항목을 비활성화시, 해당 변수가 표시되지 않습니다.

색깔 및 심볼은 채널별로 기정의 되어 있습니다.

채널 1 번 : 빨강 - 동그라미
채널 2 번 : 파랑 - 네모
채널 3 번 : 초록 - 삼각
채널 4 번 : 보라 - 다이아몬드
채널 5 번 : 검정 - 오른방향 삼각
채널 6 번 : 연두 - 왼방향 삼각
채널 7 번 : 회색 - 십자가
채널 8 번 : 오렌지 - X 글자



 : 줌(Zoom) 사용 중에 가장 왼쪽 데이타를 표시합니다.

 : 줌(Zoom) 사용 중에 전체 데이타를 표시합니다.

 : 줌(Zoom) 사용 중에 가장 오른쪽 데이타를 표시합니다.

 : 레코드 파일을 로딩합니다. 본 레코드 창을 사용하기 위한 첫 동작입니다.

 : 화면을 그래픽 파일(bmp, jpg, png 형식 지원)로 저장하거나, 데이타를 텍스트 파일(csv 형식, 추후 엑셀로 처리 편리)로 저장합니다.

유용한 기능들

- Plot 창과 동일하게 그래프를 유용하게 사용하는 방법에 대해서는 링크 를 확인하세요.

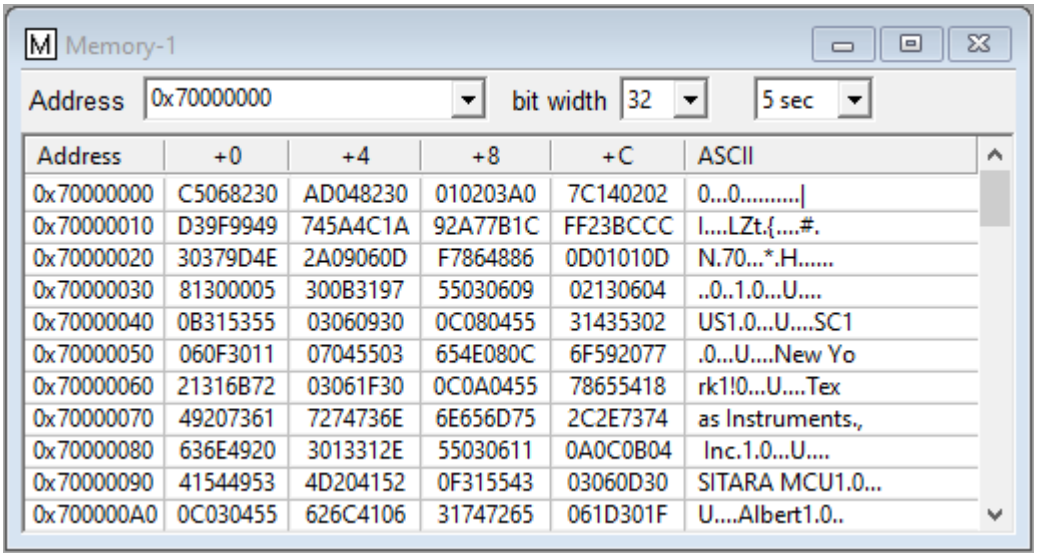
9.6 Memory

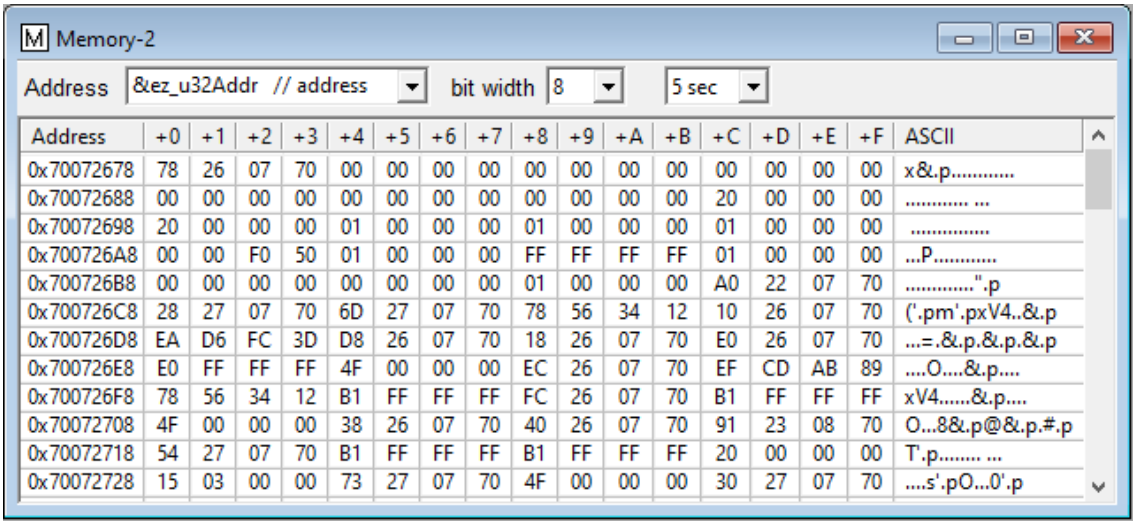
공통 사항

플래시/램 영역의 내용을 모니터링 또는 변경합니다. 단, 플래시 영역은 변경이 불가합니다.
MCU 의 통신 부하를 최소화하기 위해 보이는 항목에 대한 값만 갱신됨에 유의하세요. 또한 윈도우 크기가 크면 표시되는 데이타 량이 많기에 그만큼 easyDSP 통신에 부담을 줄 수 있으므로 필요 이상 윈도우 크기를 늘리지 않도록 부탁드립니다.
easyDSP 는 MCU 종류에 따라 모니터링 가능한 주소 범위를 제한하고 있습니다. 제한된 메모리 영역은 억세스되지 않으며 '-'로 표기됩니다.

주의사항 :
1. easyDSP 에 의해 제한되어 있지 않은 메모리 영역이라도 MCU 종류, 보안 설정에 따라 유효하지 않은 주소를 억세스할 경우, MCU 가 오동작(예 : 하드폴트) 할 수 있으므로 주소 설정에 주의하시기 바랍니다.

2. 일부 STM32 에서 Secure MPU 가 설정되어 있는 경우, 메모리 억세스시 MCU 오동작이 발생할 수 있습니다.





메모리 내용을 16 진수로, 8/16/32 비트 단위로 표시할 수 있습니다.

데이타를 변경하기 위해서는 먼저 해당 행을 선택한 후 변경하고자 하는 데이타를 마우스 좌클릭해야 합니다.

다양한 주소 입력 및 주석(//) 사용이 가능합니다. 예를 들어, '0x1234', '1234' (0x 가 생략된 hex), '&변수이름', '0x1234 // 주석'의 형식이 가능합니다.

주소 박스에 최근 사용된 값이 등록되어 있으므로 손쉽게 변경 가능하며, 최대 1kB (0x400)의 주소 영역이 한 창에 표시됩니다.

단, 주기별 데이타 업데이트는 보이는 윈도우 영역에만 국한됩니다.

주의 사항 :

1. TI C28x 코어에서는 4 바이트 얼라인된 값으로 시작 주소가 표시됩니다.

   예) 입력된 주소가 0x**0 또는 0x**1 인 경우, 0x**0 번지부터 표시

   예) 입력된 주소가 0x**2 또는 0x**3 인 경우, 0x**2 번지부터 표시

2. Arm 코어에서는 8 바이트 얼라인된 값으로 시작 주소가 표시됩니다.

예) 입력된 주소가 0x**0 ~ 0x**7 인 경우, 0x**0 번지부터 표시

예)입력된 주소가 0x**8 ~ 0x**F 인 경우, 0x**8 번지부터 표시

3. 즉, Address 박스에 입력된 주소가 메모리창에 첫번째로 표시되지 않을 수 있습니다.

4. 시작 번지 기준으로 1kB 영역만 표시됩니다.

5. 주소 입력시 변수 주소 형식을 (예 : &var) 사용한 경우, 변경된 출력 파일로 새로 MCU 를 부팅한 경우 해당 변수 주소가 변경되었다면, 메모리 윈도우의 주소도 자동 변경됩니다.

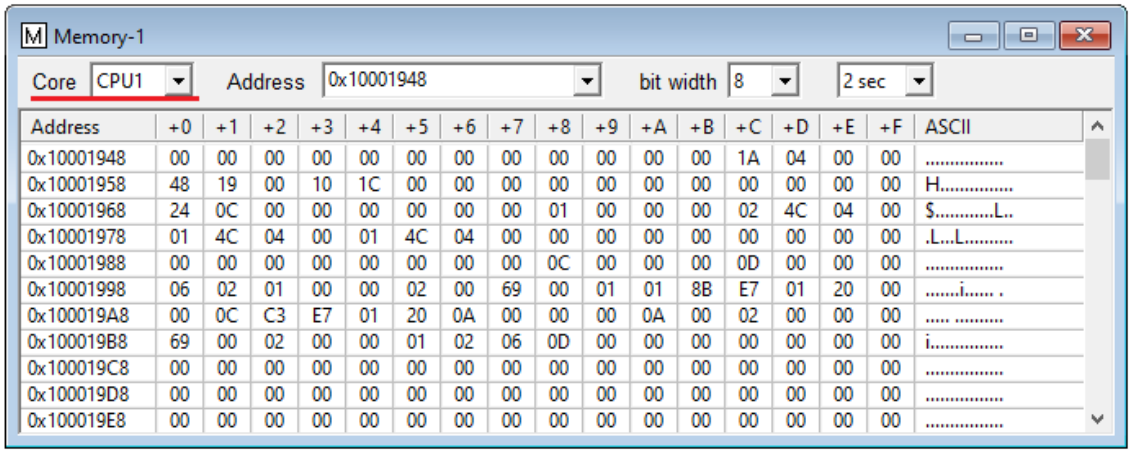
ARM 계열 멀티 코어 MCU 에서 easyDSP 가 여러 코어랑 통신할 경우

어느 코어가 메모리를 억세스할 지 지정할 수 있습니다.

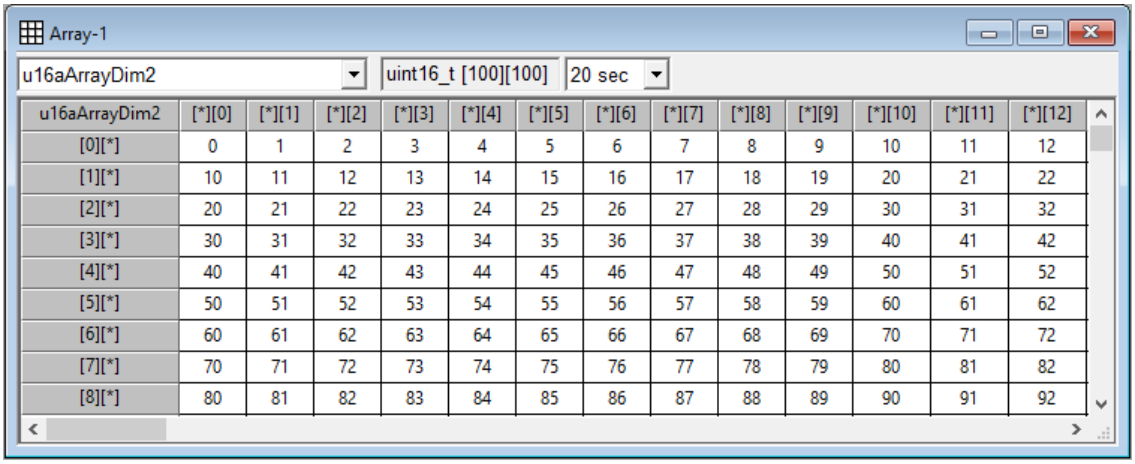
이러한 기능은 MCU 코어마다 다른 메모리 내용을 가질 경우에 유용합니다.

만약 '&n:변수' 형식으로 시작 주소를 지정할 경우, CPUn 기준으로 코어가 고정되며,

'&n:변수' 형식이 아닌 다른 형식으로 시작 주소를 입력할 때, 고정된 코어가 해제됩니다.



9.7 Array



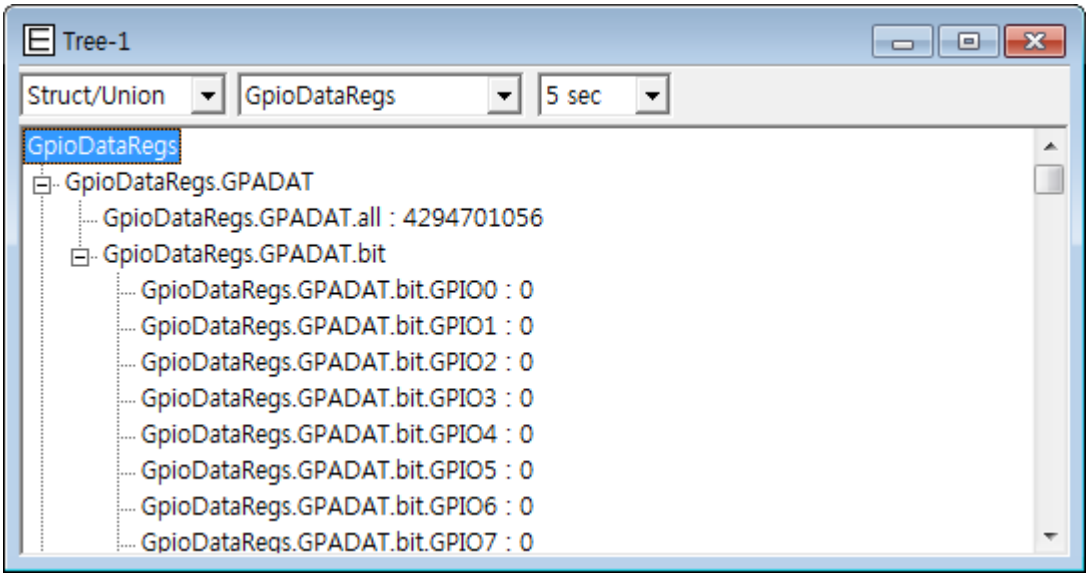
본 창에서는 1 차원 또는 2 차원 배열변수를 격자 구조로 표시합니다.

MCU 의 통신 부하를 최소화하기 위해 보이는 항목에 대한 값만 갱신됨에 유의하세요.

본 창에서 지원하는 배열 변수는 기본형식을 가지고 있어야 합니다. 구조체 등의 배열 변수는 Tree 창에서 처리하세요. 마우스 왼쪽 버튼 또는 엔터 키를 누름으로써 배열 내 특정 위치의 값을 변경할 수 있습니다.
해당 열 또는 항이 표시되는 곳을 클릭하면 해당 열, 또는 항이 모두 선택됩니다. 또한 변수 이름을 클릭하면 모든 내용이 선택됩니다. 이 선택된 내용을 Copy-Paste 하여 사용할 수 있습니다. 특히 데이타 내용을 Excel 로 Copy-Paste 하여 유용하게 사용할 수 있습니다. 단, 선택된 셀 영역이 보이지 않아서 그 값이 없는 경우라면 Copy 하기 전에 셀 값을 읽어서 사용하게 되므로 추가적인 통신 시간이 소요될 수 있습니다.

9.8 Tree

Tree 윈도우



Tree 창에서는 배열, 구조체 변수의 값을 트리 구조로 표시합니다.
MCU 의 통신 부하를 최소화하기 위해 보이는 항목에 대한 값만 갱신됨에 유의하세요.
마우스 왼쪽 버튼 클릭 또는 엔터키 입력으로, 해당 변수값을 변경할 수 있습니다.
마우스 오른쪽 버튼 클릭으로 해당 변수값의 진수 표시를 절환할 수 있습니다.

10. 문제 해결

10.1 공통 문제 해결

문제 해결 (공통)

문제 : 첫 적용시 easyDSP 통신 실패

원인 : 여러가지 이유로 인해 easyDSP 통신 실패가 가능합니다. 하기 체크 포인트를 확인하세요.

체크 포인트 1 : 램부팅 또는 플래시 프로그래밍이 정상적으로 수행되지 않는다면 하드웨어 설정을 확인하시기 바랍니다.

특히 커넥터 결선이 제대로 되어 있는지, 커넥터, 케이블에 접촉 불량이 없는지를 확인해주세요.

디버거로 MCU 를 동작시키고, 모니터링이 되는지 (즉, 하드웨어 및 소프트웨어 설정이 제대로 인지) 확인해주세요.

체크 포인트 2 : easyDSP 제공 소스파일 및 헤더 파일이 사용자 프로젝트에 포함되어야 합니다.

체크 포인트 3 : easyDSP 제공 헤더 파일에서 #define 전처리기 변수를 시스템에 맞게 설정해야 합니다.

체크 포인트 4 : main.c 에서 easyDSP 통신에 필요한 함수가 호출되어야 합니다.

체크 포인트 5 : easyDSP 제공 헤더 파일의 통신 속도와 easyDSP 프로젝트에서 설정된 통신 속도가 서로 일치해야 합니다.

체크 포인트 6 : easyDSP 에서 사용하는 통신 포트 (SCI, UART)가 사용자 프로그램에 의해 다른 GPIO 에 할당되어 있지 말아야 합니다.

체크 포인트 7 : easyDSP 에서 사용하는 GPIO 포트가 사용자 프로그램에 의해 다른 용도로 설정되어 있지 말아야 합니다.

체크 포인트 8 : easyDSP 통신을 수행하는 ISR(Interrupt Service Routine)에 충분한 시간 리소스가 할당되어야 합니다. 하기 참조 바랍니다.

문제 : 통신 시간 부족으로 인한 통신 오류

해결 : easyDSP 는 MCU 의 통신 인터럽트(우선순위는 하위)를 이용하여 통신합니다. 만약 사용자 프로그램 동작 중 상위 인터럽트 루틴 (ex, 타이머 인터럽트)에서 절대적인 시간을 많이 차지할 경우,
하위 인터럽트가 수행되는 시간이 부족하면 상기 현상이 발생됩니다. easyDSP 의 통신 절차는 하기와 같습니다.

step 1 : PC 가 주어진 bps 로 문자(커맨드)를 MCU 에 보냄. 그리고 PC 는 일정 시간 동안 대기함. (메뉴의 wait more time 은 이 대기하는 시간을 연장함)

step 2 : MCU 는 PC 로부터 전송된 문자를 해석하여 어떤 지령을 내리는 지를 파악(ex, read)한 후, 이에 맞게 PC 에 다시 문자를 보냄.

step 3 : 대기하였던 PC 는 대기 시간이 완료된 후, MCU 로부터 전달된 문자를 읽어서 화면에 표시

하위 인터럽트 수행 시간이 짧아지게 되면, Step2 에서 MCU 가 제대로 동작을 완료하기도 전에 step3 에서 PC 가 처리하게 됩니다. 따라서 통신이 깨지는 것입니다.

해결 방법으로 하기 2 가지를 적절히 사용하시기 바랍니다.

1. Wait more time 을 더 길게 설정한다.
2. 통신 BPS 를 낮춘다 (이로 인해 step 1 에서 PC 가 대기하는 시간이 길어지게 됩니다).
3. 통신하는 변수 개수를 최소화 (예를 들어, 커맨드 윈도우만 사용)
4. 가능하다면 easyDSP 통신 인터럽트의 순위 상향

문제 : 초반에는 easyDSP 통신이 동작하는데 이후 곧 동작되지 않는다

해결 : easyDSP 는 MCU 의 최하순위 통신 인터럽트를 사용합니다. 만약 MCU 프로그램 동작 중 MCU 의 처리부분이 급증하여 본 인터럽트 루틴에 할당된 동작 시간이 부족할 경우 본 현상이 발생하게 됩니다. 따라서 충분한 시간을

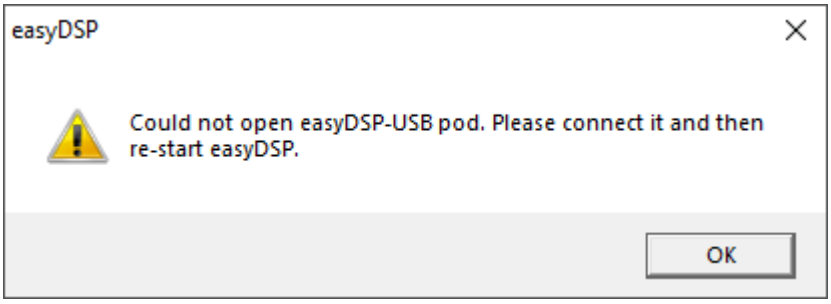
확보하여 주는 것만이 해결 방법입니다. .

해결 : 일부 시스템에서는 시스템 동작중 전력 소자 스위칭으로 인한 노이즈 발생으로 통신이 실패할 수 있습니다. 적절한 노이즈 대책을 수행하시기 바랍니다. 예를 들어 광 케이블 버전 사용.

문제 : easyDSP 연결이 안되거나 원활하지 않아서 하기 메시지 발생

원인 : 물리적 연결 오류

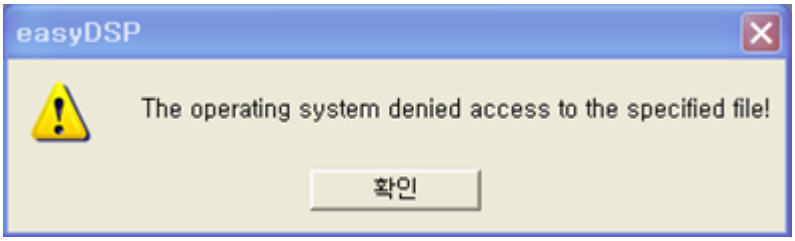
해결 : PC 와 easyDSP pod 간 직접 연결 (USB 확장 포트 사용하지 말고) 내지는 사용 USB 포트 변경 및 USB 케이블 변경



문제 : 하기와 유사한 오류 메시지 발생

원인 : 32 비트 윈도우즈 사용

해결 : 64 비트 윈도우즈 사용



문제 : 홈페이지 접속 오류

원인 : easyDSP 웹사이트에서 (www.easyDSP.com) 하루에 처리할 수 있는 트래픽 제한으로 인해 일시적으로 접속이 안되는 현상

해결 : 다음 날에 접속을 시도

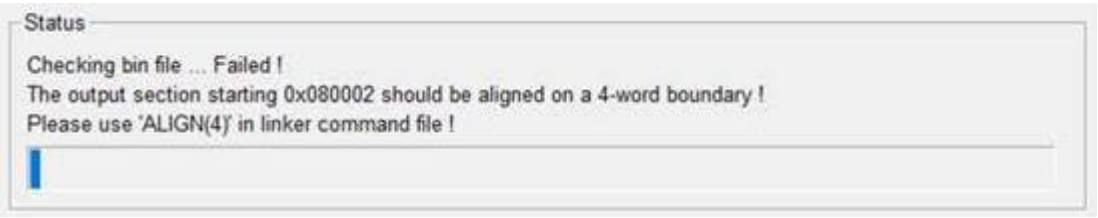
10.2 C28x 문제 해결

문제 해결 (TI C28x)

문제 : 매뉴얼에서 지정한 SCI-A 의 GPIO 포트를 사용할 수 없을 때

C28x 는 SCI-A 의 지정된 GPIO 포트에서만 램부팅을 지원합니다. 따라서 easyDSP 를 사용하여 램부팅 또는 플래시 프로그래밍을 하기 위해서는 해당 GPIO 포트의 SCIA 에 easyDSP 가 연결되어야 합니다.
만약 easyDSP 로 변수 모니터링만 수행한다면 다른 어떠한 SCI, GPIO 포트에 연결될 수 있습니다. 단, easyDSP 가 제공하는 소스파일을 해당 SCI, GPIO 포트에 맞게 수정해야 합니다.
easyDSP 로 램부팅 또는 플래시 프로그래밍을 수행하면서, 지정된 GPIO 포트를 사용하지 않는 방법은 도움말 "MCU 별 사용법>C28x>SCI-A 가 아닌 포트 사용" 을 참조하시기 바랍니다.
MCU 리셋 초기에 easyDSP 를 지정된 GPIO 포트에 연결했다가, 램부팅 및 플래시 프로그래밍을 완료한 후, 사용자 프로그램이 수행되면, easyDSP 를 다른 GPIO 포트에 연결하는 방식입니다.

문제 : 플래시롬 대화상자에서 하기 에러 메시지 발생



해결 : easyDSP 는 TI 에서 제공한 flashAPI 를 통해 플래시롬을 억세스합니다. Gen3 MCU (예 : F2807x, F28002x, F28004x, F2837x, F2738x)의 flashAPI 는 MCU 종류에 따라 섹션이 최소 4 워드 또는 권장 8 워드에 얼라인되도록 요구하고 있습니다. 즉, C28x 코어의 경우 섹션 시작 주소의 최하위 번지가 0x0, 0x4, 0x8, 0xC 로 마감되어야 하며, Arm Cortex-M4 (ex, F2838x CM)의 경우, 0x0, 0x8 로 마감되어야 합니다. 상기 그림에서는 0x2 로 마감되어 에러가 발생하였습니다. 해결 방법은 cmd 파일에서 플래시에 저장되는 섹션에 대해서 ALIGN()를 붙이는 것입니다. 아래 TI 에서 제공하는 링커 커맨드 파일에 대표적인 섹션에 대해 이미 적용되어 있음을 알 수 있습니다. 사용자가 섹션을 새로 생성할 경우에도 ALIGN(*)이 적용될 수 있도록 주의 부탁드립니다. 이와 같은 조치 이후에도 문제가 지속될 시에는 map 파일을 참조하셔서 문제를 일으키는 섹션 (상기 그림에서는 0x080002 시작주소를 가지는 섹션)이 어떤 섹션인지 파악하신 후, 해당 섹션의 시작 주소가 제대로 얼라인될 수 있도록 조치하여 주십시오.

<TMS320F280049 경우>

```
SECTIONS
{
   codestart        : > BEGIN,     PAGE = 0, ALIGN(4)
   .text            : >> FLASH_BANK0_SEC2 | FLASH_BANK0_SEC3 | FLASH_BANK0_SEC5,   PAGE = 0, ALIGN(4)
   .cinit           : > FLASH_BANK0_SEC1,     PAGE = 0, ALIGN(4)
   .switch          : > FLASH_BANK0_SEC1,     PAGE = 0, ALIGN(4)
   .reset           : > RESET,     PAGE = 0, TYPE = DSECT /* not used, */

   .stack           : > RAMM1,     PAGE = 1

#if defined(__TI_EABI__)
   .init_array      : > FLASH_BANK0_SEC1,       PAGE = 0,       ALIGN(4)
   .bss             : > RAMLS5,       PAGE = 1
   .bss:output      : > RAMLS3,       PAGE = 0
   .bss:cio         : > RAMLS0,       PAGE = 0
   .data            : > RAMLS5,       PAGE = 1
   .sysmem          : > RAMLS5,       PAGE = 1
   /* Initalized sections go in Flash */
   .const           : > FLASH_BANK0_SEC4,       PAGE = 0,       ALIGN(4)
#else
   .pinit           : > FLASH_BANK0_SEC1,       PAGE = 0,       ALIGN(4)
   .ebss            : > RAMLS5,       PAGE = 1
   .esysmem         : > RAMLS5,       PAGE = 1
   .cio             : > RAMLS0,       PAGE = 0
   .econst          : > FLASH_BANK0_SEC4,    PAGE = 0, ALIGN(4)
#endif

   ramgs0           : > RAMGS0,    PAGE = 1
   ramgs1           : > RAMGS1,    PAGE = 1
```

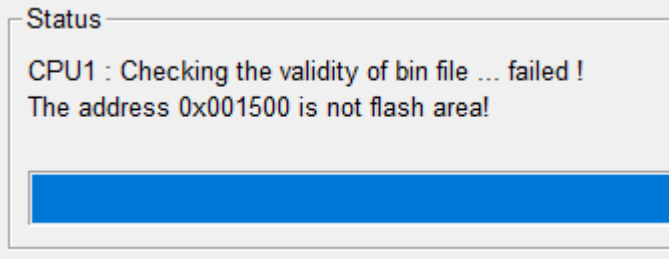
<TMS320F28388 CPU1/CPU2 경우>

```
SECTIONS
{
   codestart           : > BEGIN, ALIGN(8)
   .text               : >> FLASH1 | FLASH2 | FLASH3 | FLASH4, ALIGN(8)
   .cinit              : > FLASH4, ALIGN(8)
   .switch             : > FLASH1, ALIGN(8)
   .reset              : > RESET, TYPE = DSECT /* not used, */
   .stack              : > RAMM1

#if defined(__TI_EABI__)
   .init_array      : > FLASH1, ALIGN(8)
   .bss             : > RAMLS5
   .bss:output      : > RAMLS3
   .bss:cio         : > RAMLS5
   .data            : > RAMLS5
   .sysmem          : > RAMLS5
   /* Initalized sections go in Flash */
   .const           : > FLASH5, ALIGN(8)
#else
   .pinit           : > FLASH1, ALIGN(8)
   .ebss            : > RAMLS5
   .esysmem         : > RAMLS5
   .cio             : > RAMLS5
   /* Initalized sections go in Flash */
   .econst          : >> FLASH4 | FLASH5, ALIGN(8)
#endif
```

<TMS320F28388 CM 경우>

```
SECTIONS
{
   .resetisr        : > CMBANK0_RESETISR, ALIGN(16)
   .vftable         : > CMBANK0_SECTOR0, ALIGN(16)   /* Application placed vector table in Flash*/
   .vtable          : > S0RAM             /* Application placed vector table in RAM*/
   .text            : >> CMBANK0_SECTOR0 | CMBANK0_SECTOR1, ALIGN(16)
   .cinit           : > CMBANK0_SECTOR0, ALIGN(16)
   .pinit           : >> CMBANK0_SECTOR0 | CMBANK0_SECTOR1, ALIGN(16)
   .switch          : >> CMBANK0_SECTOR0 | CMBANK0_SECTOR1, ALIGN(16)
   .sysmem          : > S2RAM

   .stack           : > C1RAM
   .ebss            : > C1RAM
   .econst          : >> CMBANK0_SECTOR0 | CMBANK0_SECTOR1, ALIGN(16)
   .esysmem         : > C1RAM
   .data            : > S3RAM
   .bss             : > S3RAM
   .const           : >> CMBANK0_SECTOR0 | CMBANK0_SECTOR1, ALIGN(16)
```

문제 : 플래시 대화상자 진입시 The address xxx is not flash area ! 에러 메시지



원인 : 램 영역에 initialized section 이 위치하여 플래시 프로그래밍이 불가능함. 특히 CLA 프로그래밍시 램에 위치한 사용자 변수의 초기값이 설정되는 경우

해결 : 하기 매뉴얼 캡처 참조. 사용자 변수의 초기값 설정이 필요한 경우, 이를 main() 초기 부분에 변수 라이팅함으로써 해결

10.2.3 C Language Restrictions

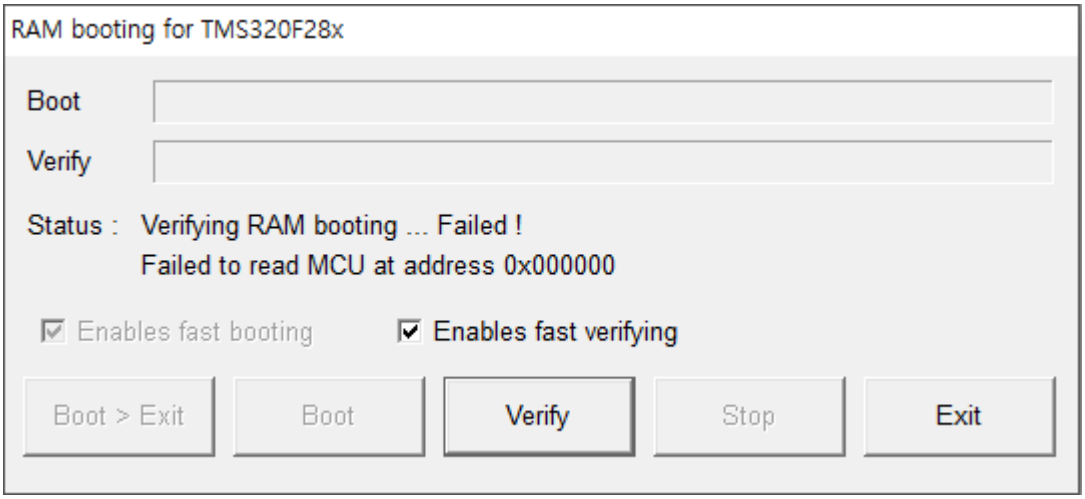
There are several additional restrictions to the C language for CLA.

- Defining and initializing global/static data is not supported.

  Since the CLA code is executed in an interrupt driven environment, there is no C system boot sequence. As a result, global/static data initialization must be done during program execution, either by the C28x driver code or within a CLA function.

  Variables defined as const can be initialized globally. The compiler creates initialized data sections named .const_cla to hold these variables.
- CLA code cannot call C28x functions. The linker provides a diagnostic message if code compiled for C28 calls code compiled for CLA or if code compiled for CLA calls code compiled for C28.
- Recursive function calls are not supported.
- The use of function pointers is not supported.

문제 : 램부팅은 성공하나 0 번지부터 읽기 실패로 베리파이 실패



verify 작업은 easyDSP 와 MCU 간 통신에 의해 진행됩니다. 따라서 통신을 방해하는 모든 요소가 원인이 될 수 있습니다.

원인 1 : easyDSP 제공 통신용 소스파일이 프로젝트에 포함되지 않음.
해결 1 : 도움말을 참조하여 easyDSP 제공 통신용 소스파일을 프로젝트에 포함하고 적절한 설정

원인 2 : 사용자 프로그램에서 easyDSP 가 사용하는 GPIO 에 대해 설정
해결 2 : 해당 설정 제거

원인 3 : easyDSP 통신을 위한 적절한 리소스가 할당되지 못함.
해결 3 : 만약 ePWM 인터럽트 주파수가 매우 높다면 이를 적절히 낮춰 easyDSP 통신 인터럽트에 충분한 시간을 할당함.

문제 : controlSUITE 사용시 easy28x_xx.c 소스파일에서 컴파일 에러 발생

원인 : controlSUITE 사용시 정의된 레지스터 이름이 달라 컴파일 에러 발생
해결 : C2000Ware 사용

문제 : 변수 개수를 증가시키거나 매우 큰 사이즈의 배열을 설정할 때 MCU 가 정상동작하지 않음

원인 : 이전 버전의 TI 소스파일의 버그
해결 : 최신 버전의 TI 소스파일 (C2000Ware) 사용

문제 : F2838x 가 동작하지 않음

원인 : C2000Ware_3_02_00_00 버전부터 TI 제공 소스파일의 기본 클럭주파수가 20MHz 에서 25MHz 로 변경되었는데 실제 보드에서는 클럭 주파수 20MHz 사용중.
해결 : USE_20MHZ_XTAL 를 predefine 하여 비트필드라면 f2838x_sysctrl.c 파일내에서, DriverLib 라면 device.h 파일에서 클럭 주파수가 제대로 설정되도록 조치 바랍니다. 하기 TI 문서

(C2000Ware_3_02_00_00 release note) 참조하세요.

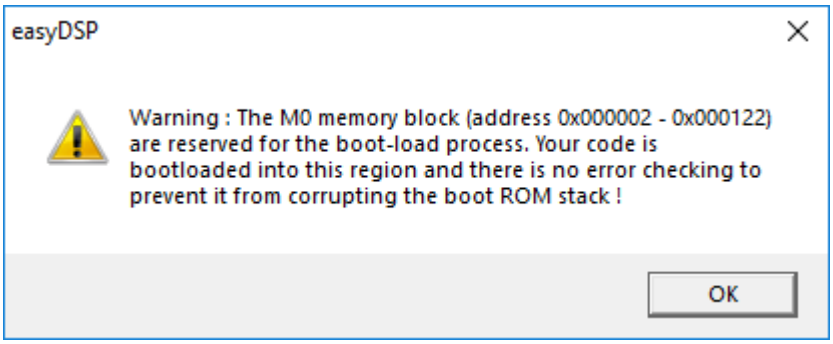
- F2838x driverlib and examples updated to use 25MHz XTAL clock as default input clock

  Note: By default, Device_init function in driverlib and InitSysctrl function in bitfield examples assumes that the XTAL frequency is 25MHz. If a 20MHz XTAL is used, please add a predefined symbol "USE_20MHZ_XTAL" in your CCS project. If a different XTAL is used, you need to update the PLL multipliers and dividers accordingly. Note that the latest F2838x controlCARDs (Rev.B and later) have been updated to use 25MHz XTAL by default. If you have an older 20MHz XTAL controlCARD (E1, E2, or Rev.A), refer to the controlCARD documentation on steps to reconfigure the controlCARD from 20MHz to 25MHz.

문제 : 램 부팅 초기에 하기의 경고 메시지 출현

해결 : 램 부팅시 부트 롬 프로그램이 사용하는 램 영역이 사용자에 의해 사용되지 말아야 하는 데, 사용자 프로그램이 이 영역을 사용하는 것에 대해서 경고를 띄우는 것입니다.

예를 들어, 하기 메시지는 28377 사용시 사용자 프로그램이 0x2 - 0x122 사이에 존재하기 때문에 발생된 경고입니다.



TI 의 Technical Reference Manual (Literature Number: SPRUHM8I, Revised September 2019)에 해당 영역은 사용자가 사용하지 말아야 할 영역으로 나와 있습니다.

Table 4-19. Reserved RAM and Flash Memory Map for CPU1

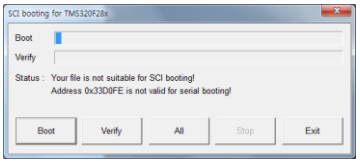
Memory	Description	Start Address	End Address	Length
RAM	Boot ROM	0x0000 0002	0x0000 0122	0x0121
	TI-RTOS(1)	0x0000 0780	0x0000 07FF	0x0080
Flash	TI-RTOS(1)(2)	0x0008 2000	0x0008 2823	0x0824

(1) If the user is not planning on using TI-RTOS in ROM, then these memory locations are free to be used by the application.
(2) For using the TI-RTOS in flash sector A, TI recommends that this sector be made unsecure, or at minimum, the sector should be verified that there is no secure zone claiming this sector.

사용하시는 커맨드 파일을 적절히 수정하셔서 본 영역이 사용되지 않도록 수정하여 주십시오.

문제 : 하기 메시지로 램 부팅이 실패한다

해결 : 링크파일에서 프로그램을 플래시 영역에 할당했기 때문에 램 부팅이 안되는 것입니다. 메시지의 메모리 주소는 플래시 영역입니다. 따라서 링크파일을 변경하여 모든 메모리 영역을 램에 할당한 후 램 부팅을 시도하십시요.



문제 : Auto Bauding 실패

해결 : 이 문제는 대부분 잘못된 하드웨어 연결(easyDSP pod 와 MCU 보드간)에 의한 것입니다.

스텝 1 : 도움말을 기반으로 현재 연결이 올바른 지 다시 한번 확인해 주십시요. 만약 오류가 발견되지 않는 다면 스텝 2 로 넘어갑니다.

스텝 2 : 아래 그림을 참조하여 부팅시 easyDSP 핀의 실제 파형을 관찰하여 주십시요. easyDSP 의 /RESET 신호가 MCU 에 직결되지 않을 경우에는 (예를 들어, 전원 감시 IC 가 그 중간에 위치할 경우), DSP 의 리셋 신호도 같이 관찰하여 주십시오.

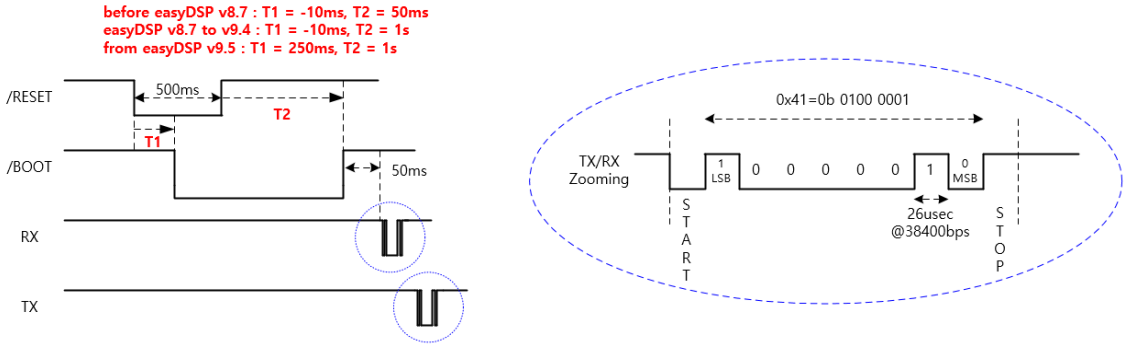
확인 1 : easyDSP 의 /BOOT 신호가 HIGH 가 되기 전에, MCU 리셋 입력 신호(XRS)가 HIGH 로 되어야 함.

easyDSP /RESET 신호가 DSP 에 직결된 경우에는 하기 그림처럼 이 조건이 보장됨.

직결되지 않고 중간에 전원 감시 IC 가 있을 경우에는 전원 감시 IC 의 동작 조건에 따라 이 조건이 만족되지 않을 수 있음.

확인 2 : 0x41 의 RX 가 나온 뒤 일정시간 뒤 동일값의 TX 가 나옴

/BOOT 핀이 HIGH 로 올라간 이후 50msec 이후에 RX 를 통해 easyDSP 로부터 MCU 에게 0x41 이 전달됩니다. 보딩 bps 는 DSP 종류, SCI 부팅 속도 설정에 따라 다를 수 있습니다. 전달된 0x41 을 기반으로 MCU 는 bps 를 파악하여, TX 를 통해 파악된 bps 로 다시 easyDSP 에 보내줍니다. 이러한 작업을 통해, 상호간에 bps 를 공유하는 auto bauding 이 완료되는 것입니다.



문제 : CCS 상에서 빌드를 하고 나면 아래와 같은 에러가 발생

```
undefined first referenced
 symbol        in file
--------- ----------------
LL$$OR     C:₩₩tidcs₩₩c28₩₩DSP2833x₩₩Project₩₩Debug₩₩easy2833x_sci_v7.3.obj
ULL$$CMP   C:₩₩tidcs₩₩c28₩₩DSP2833x₩₩Project₩₩Debug₩₩easy2833x_sci_v7.3.obj

error: unresolved symbols remain
error: errors encountered during linking; "./Debug/inverter.out" not built

>> Compilation failure
```

해결 : MCU 는 일부 연산을 런타임라이브러리를 통해 지원합니다. 상기 에러에서
"ULL$$CMP" = unsigned long long comparison
"LL$$OR" = long long oring
을 각각 의미합니다. 컴파일시 런타임라이브러리를 포함 하십시오.

문제 : 모든 변수 형식이 'int'로 표기

원인 : 프로젝트 설정에서 debugging model 을 잘 못 지정한 경우입니다 (dwarf 형식으로 컴파일된 out 파일을 coff 형식으로 지정)

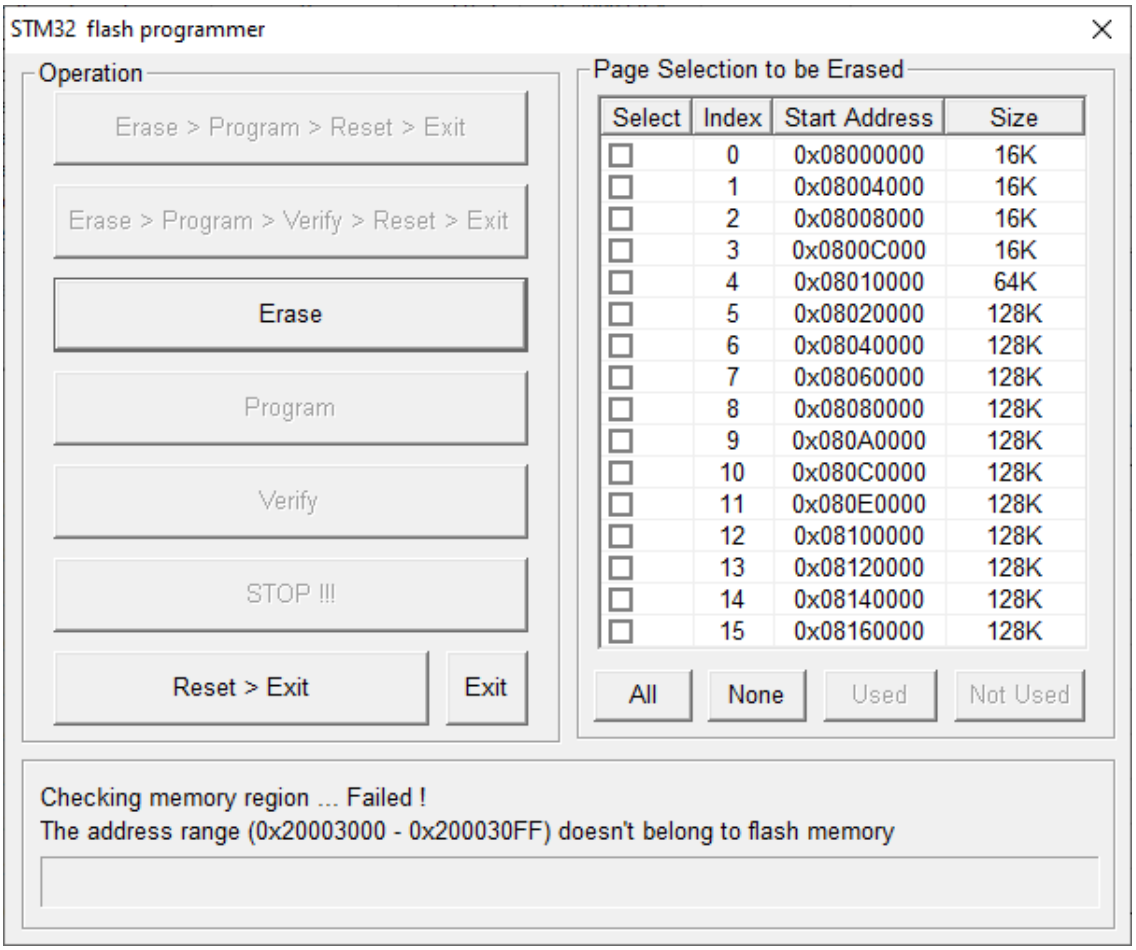
해결 : 최신 easyDSP 프로그램을 사용하시고 프로젝트 설정에서 debugging model 을 올바르게 설정

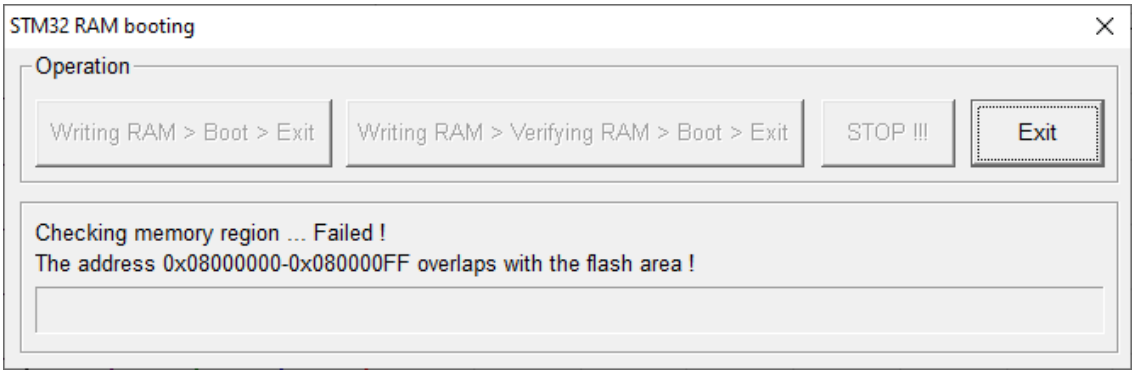
10.3 STM32 문제 해결

문제 해결 (ST STM32)

문제 : 플래시롬 대화상자에서 또는 램부팅 대화상자에서 하기 오류 메시지

해결 : 사용자 코드가 플래시 기반일 경우에만 플래시롬 대화상자에서 처리가능하며, 램 기반일 경우에만 램부팅 대화상자에서 처리 가능합니다. 기타의 경우에는 처리할 수 없습니다.





문제 : 부트로더 진입 실패

해결 : 이 문제는 대부분 잘못된 하드웨어 연결(easyDSP pod 와 MCU 보드간)에 의한 것입니다.
스텝 1 : 도움말을 기반으로 현재 연결이 올바른 지 다시 한번 확인해 주십시요. 만약 오류가 발견되지 않는 다면 스텝 2 로 넘어갑니다.
스텝 2 : 아래 그림을 참조하여 부팅시 easyDSP 핀의 실제 파형을 관찰하여 주십시요. easyDSP 의 /RESET 신호가 DSP 에 직결되지 않을 경우에는 (예를 들어, 전원 감시 IC 가 그 중간에 위치할 경우), DSP 의 리셋 신호를 직접 관찰하여 주십시오.

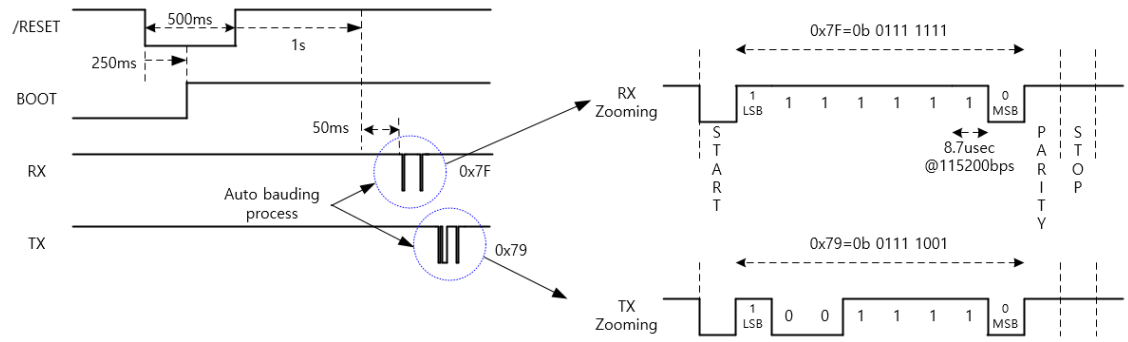
확인 1 : MCU 리셋 입력 신호(NRST)가 Low->High 로 변경될 때, easyDSP 의 BOOT 신호가 High 이어야 합니다.

easyDSP /RESET 신호가 MCU 에 직결된 경우에는 하기 그림처럼 이 조건이 보장됨.

직결되지 않고 중간에 전원 감시 IC 가 있을 경우에는 전원 감시 IC 의 동작 조건에 따라 이 조건이 만족되지 않을 수 있음.

확인 2 : easyDSP RX 핀에 0x7F 값이 송출된 후 TX 핀에 0x79 값이 송출됨.

/RESET 핀이 High 가 된 이후 약 1.050sec 이후에 RX 를 통해 easyDSP 로부터 MCU 에게 0x7F(even parity)가 전달됩니다. 보딩 bps 는 보통 115200bps 또는 57600bps 이지만 MCU 종류에 따라 다를 수 있습니다. 전달된 0x7F 을 기반으로 MCU 는 bps 를 파악하여, TX 를 통해 파악된 bps 로 다시 easyDSP 에 0x79(even parity)값을 보내줍니다. 이러한 작업을 통해, 상호간에 bps 를 공유하는 auto bauding 이 완료되는 것입니다.
오토보딩이 실패할 수 있는 경우는 하기와 같습니다.
경우 1 : 0x79 가 관찰되지 않은 경우. 이 경우 보통 Option Byte 를 제대로 설정하지 않아서 리셋 이후 MCU 가 부트로더로 진입하지 않는 경우이므로 Option Byte 를 재 설정하시기 바랍니다.
경우 2 : 0x79 가 관찰되나 0x7F 와 다른 bps 를 보이는 경우. 이 경우는 MCU 가 정확한 bps 로 오토보딩하지 못한 예이며, easydsp@gmail.com 으로 리포팅하여 주시기 바랍니다.



11. 유용한 팁

11.1 DA 컨버터 사용법

MCU 보드내에 DA 컨버터가 있을 경우, 사용자는 DSP 프로그램의 변수값을 DA 컨버터를 통해 출력함으로써 오실로스코프와 같은 계측기로 관찰할 수 있습니다.
easyDSP 를 사용하므로써 DA 컨버터의 출력 내용(즉, 변수)을 온라인으로 용이하게 변경하는 방법을 설명합니다.

Step 1 : da.h 파일의 수정

easyDSP 는 DA 컨버터의 제어를 위해 2 개의 MCU 프로그램 파일(da.c, da.h)을 제공합니다. 우선 da.h 를 살펴보면 다음과 같습니다.

```
// File name : da.c
// function : DA output control

// variable explanation(#=1,2,3,4)
// da# : address of variable
// da#_type = 0 ; the variable is float
// = 1 ; the variable is integer
// da#_mid : mid value
// da#_rng : da scale

// use this routine in EasyDSP as below
// da1=&var_float
// da1_type=0
// da1_mid=0.
// da1_rng=20
// da2 = &var_int
// da2_type = 1
// da2_mid= 0.
```

```
// da2_rng = 20

#ifndef _DA_EasyDSP
#define _DA_EasyDSP

// you should specify the da address of your own
#define DA1_ADDR (*(int *)0X03C000e)
#define DA2_ADDR (*(int *)0X03C000d)
#define DA3_ADDR (*(int *)0X03C000b)
#define DA4_ADDR (*(int *)0X03C0007)
#define

extern unsigned int da1, da2, da3, da4, da1_type, da2_type, da3_type, da4_type;
extern float da1_rng, da1_val, da1_mid;
extern float da2_rng, da2_val, da2_mid;
extern float da3_rng, da3_val, da3_mid;
extern float da4_rng, da4_val, da4_mid;

// 12 bit DA
#define DA12(num) ₩
da##num##_val = (da##num##_type == 0 ? *(float *)da##num : (float)(*(int *)da##num)) ; ₩
DA##num##_ADDR = (int)((da##num##_val-da##num##_mid )* 0x7ff/da##num##_rng) + 0x800 ;

// 8 bit DA
#define DA8(num) ₩
da##num##_val = (da##num##_type == 0 ? *(float *)da##num : (float)(*(int *)da##num)) ; ₩
DA##num##_ADDR = (int)((da##num##_val-da##num##_mid )*0x7f/da##num##_rng) + 0x80 ;



#endif
```

우선 DA#_ADDR 정의 구문에서 당신의 DSP 보드의 DA 번지를 설정하십시요(#= 1,2,3,4).그 후, 당신의 DA 컨버터에 따라 적절한 DA 출력용 매크로 함수를 정의합니다. < BR> 위 코드에서는 8bit 또는 12bit 양방향(+/-) DA 컨버터의 예제를 표시합니다. 각 변수는 da.c 파일에 정의되어 있으며 그 의미를 살펴보면 다음과 같습니다.

da# = DA 채널 #의 값. 즉, DA 출력을 하고자하는 변수의 번지

da#_type = da#에서 번지값이 설정된 변수의 타입. 정수일 경우 1, 실수일 경우 0

da#_rng = 표시값의 범위

da#_mid = 표시값의 중간값

da#_val = 채널#에 라이트할 값

Step 2 : MCU 프로그램 수정

우선 da.c 의 컴파일한 파일인 da.obj 를 linker 에 추가하여 MCU 프로그램을 생성합니다. 그 후, DA 출력을 원하는 파일의 첫부분에 da.h 를 include 한 후, DA 출력을 원하는 위치에 다음과 같이 da.h 에서 정의된 DA 출력용 매크로 함수를 적습니다.

```
#include "da.h"

.....

DA12(1);
DA12(2);
DA12(3);
DA12(4);
.....
```

Step 3 : easyDSP 사용

easyDSP 내의 커맨드 윈도우내에서 다음과 같은 형식으로 DA 출력을 제어할 수 있습니다.

da1=&var_float
da1_type=**0**
da1_mid=0
da1_rng=20
da2 = &var_int
da2_type = **1**
da2_mid= 0
da2_rng = 20

주의점 : DA 출력을 위한 매크로중 나누기 연산은 수행시간을 저하시킵니다. 따라서 고속의 DA 출력을 원할 경우, 나누기 연산 부분을 적절한 곱셈으로 대치하여 사용하세요.

11.2 기타

기타 팁

---

easyDSP 로 플래시만 구울 경우 NEW

easyDSP 의 각종 통신 기능은 사용하지 않고, 단순히 주어진 출력 파일(예: *.out 파일)을 MCU 플래시에 프로그래밍할 용도로만 easyDSP 를 사용한다면, 해당 출력 파일을 사용하여 easyDSP 프로젝트를 생성한 후 플래시롬

대화상자로 이동하여 플래시를 프로그래밍하면 됩니다.
각종 통신 기능을 사용할 때 필요한 통신용 소스파일 추가하는 등등의 작업은 필요가 없습니다.

커맨드실행 없이 한 줄 삽입

커맨드 창에서의 엔터키 입력은 해당라인의 커맨드를 실행하게 합니다. 커맨드 실행 없이 단순히 한 줄을 삽입하고 싶을 시에는 컨트롤 키와 엔터키를 동시에 누르세요. 이것은 또한  버튼을 클릭하는 것과 동일합니다.

커맨드창에서 변수 변경 후 바로 값 확인

변수 값을 바꾼 후 실제 변수 값을 확인하고 싶으면, 해당 라인에 커서를 위치시킨 후 마우스 오른쪽 버튼을 클릭하십시오. 마우스 오른쪽 버튼은 'update'  버튼과 동일한 기능이므로, 커맨드 윈도우의 내용을 블럭으로 선택한 후 사용할 수도 있습니다.

플래시 프로그램 및 환경변수 저장

플래시의 마지막 섹터는 프로그램의 환경변수 저장용으로, 나머지 섹터는 MCU 프로그램 자체로 사용하는 것이 편리합니다. 이 경우 Sector Erase 기능을 사용하여 선택적으로 섹터를 지운 후, 프로그램 코드만을 필요한 섹터에 프로그래밍할 수 있습니다.

11.3 자주 묻는 질문들

easyDSP 는 JTAG/SWD 디버거와 어떻게 다른 것인지?

디버거와 easyDSP 는 다른 용도를 가지고 있습니다.
디버거는 break point 및 step 별 진행으로 초기 개발단계에서 보드 및 소프트웨어의 검증에 유용합니다. 하지만 어떤 어플리케이션에는 (ex, 모터 드라이브) 시스템 운전중 이러한 기능은 사용할 수 가 없습니다.
결국 운전중인 시스템에서는 사용자코드 변수를 모니터링하여 디버깅하게 됩니다.
디버거를 사용하여 변수를 모니터링하게 되면, 모니터링 할 수 있는 변수 개수 및 속도에 제약이 있으며,
또한 디버거는 노이즈 내성이 약하여, 고전압/고전류 시스템의 경우, 시스템 동작과 더불어 디버거 연결이 끊기는 경우가 자주 발생됩니다.
또한 제품 양산시 IP 보호를 위해, 디버거의 동작을 제한하기도 합니다.
반면에 easyDSP 는 SCI/UART 통신 포트를 사용하기에 노이즈 내성이 강하여 고전압/고전류 시스템 모니터링에 적합합니다.
필요에 따라 easyDSP 와 디버거를 각각 내지 같이 사용하면 최상의 디버깅 환경을 구현할 수 있습니다.

easyDSP 의 변수 읽기의 신뢰성은?

읽은 변수 값의 정확성은 보장되지 않습니다. 즉, 도중에 오류가 발생하여 잘못된 변수 값을 읽을 수도 있습니다. 사용자의 주의를 부탁 드립니다.

easyDSP 의 변수 쓰기의 신뢰성은?

변수에 쓰기 전에 2 바이트 체크섬 형식으로 그 내용을 확인하므로 대부분의 경우에는 안전성이 확보됩니다. 하지만 매우 적은 확률로 잘못된 값이 쓰여질 수 있습니다.

easyDSP 가 통신 실패시 표시하는 변수값은?

노이즈에 의한 통신의 불안정 또는 시리얼 통신에 할당된 시간 리소스가 부족함에 따른 통신 시퀀스의 불안 등이 통신 실패의 원인이 될 수 있으며, 이 경우 '?'가 표시됩니다. 플롯창 또는 차트창의 경우에는 아무 것도 표시되지 않게 됩니다.

플래시 프로그래밍은 믿을 만한가?

'Program' 버튼을 클릭함으로써는 플래쉬 롬에 정확한 값이 라이트 된 것인지는 확인할 수 없습니다. 따라서 반드시 'Program'후에 'Verify'버튼을 클릭해주세요.

컴파일이나 링킹도 모두 easyDSP 내에서 하는지?

아닙니다. 칩 제조사의 개발 환경 (IDE)를 사용하세요.

easyDSP 에서 직접 변수를 관측하는데 관측할 수 있는 용량은 얼마인지?

PC 속도/메모리가 허용하는 한 무한대입니다 .

12. Driver 설치방법

12.1 설치 방법

윈도우즈 OS 종류	드라이버 설치 방법
Windows 11 Windows 10 Windows 8.1 Windows 8 Windows 7	easyDSP 사용전, easyDSP 실행 폴더의 'Driver' 폴더 안에 있는 " CDM212364_Setup.exe "를 실행합니다. easyDSP 를 PC 에 연결하지 않은 상태에서 진행하십시오 .최신 드라이버 파일은 http://www.ftdichip.com/Drivers/D2XX.htm 에서 다운받으실 수도 있습니다. 자세한 과정은 Windows 7 의 경우를 참조 하십시오. 기타의 OS 에서도 유사한 과정이 사용됩니다 . 드라이버 설치시 문제가 발생한 경우에는 아래 링크를 참조 하세요 . Windows 10/11 설치 가이드 Windows 8 설치 가이드 Windows 7 설치 가이드
Windows Vista Windows XP	easyDSP 설치 폴더에는 포함되어 있지 않습니다. 다운로드하셔서 사용하시기 바랍니다. 드라이버 파일 : http://www.ftdichip.com/Drivers/CDM/CDM20824_Setup.exe

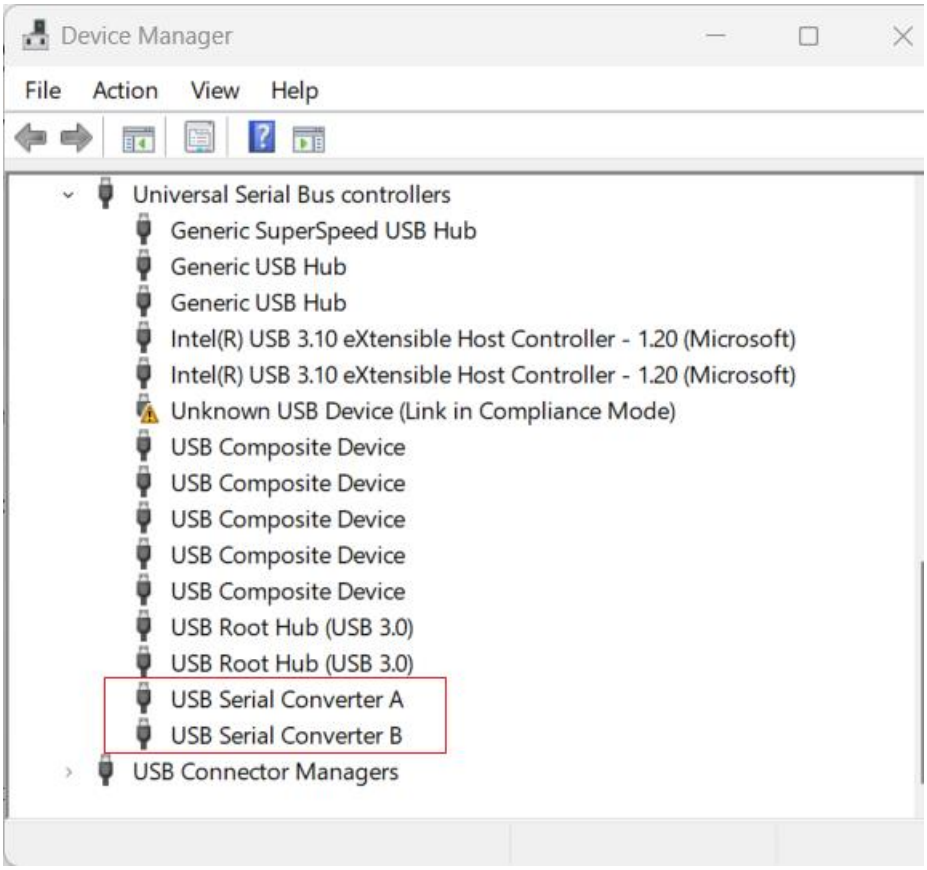
	설치 방법 : Windows Vista 설치 가이드 Windows XP 설치 가이드

easyDSP 는 FTDI 사의 FT2232 USB 컨트롤러 칩을 사용하고 있으므로 해당 드라이버 설치 파일 및 방법은 FT2232 의 D2XX direct driver 와 동일합니다. 아래 페이지에서 최신 드라이버 파일 및 설치 가이드를 참조하실 수 있습니다 .

http://www.ftdichip.com/Drivers/D2XX.htm

http://www.ftdichip.com/Documents/InstallGuides.htm

드라이버가 잘 설치되면 easyDSP 포드를 컴퓨터에 연결할 때 장치 관리자에 USB Serial Converter A/B 가 표기됩니다.

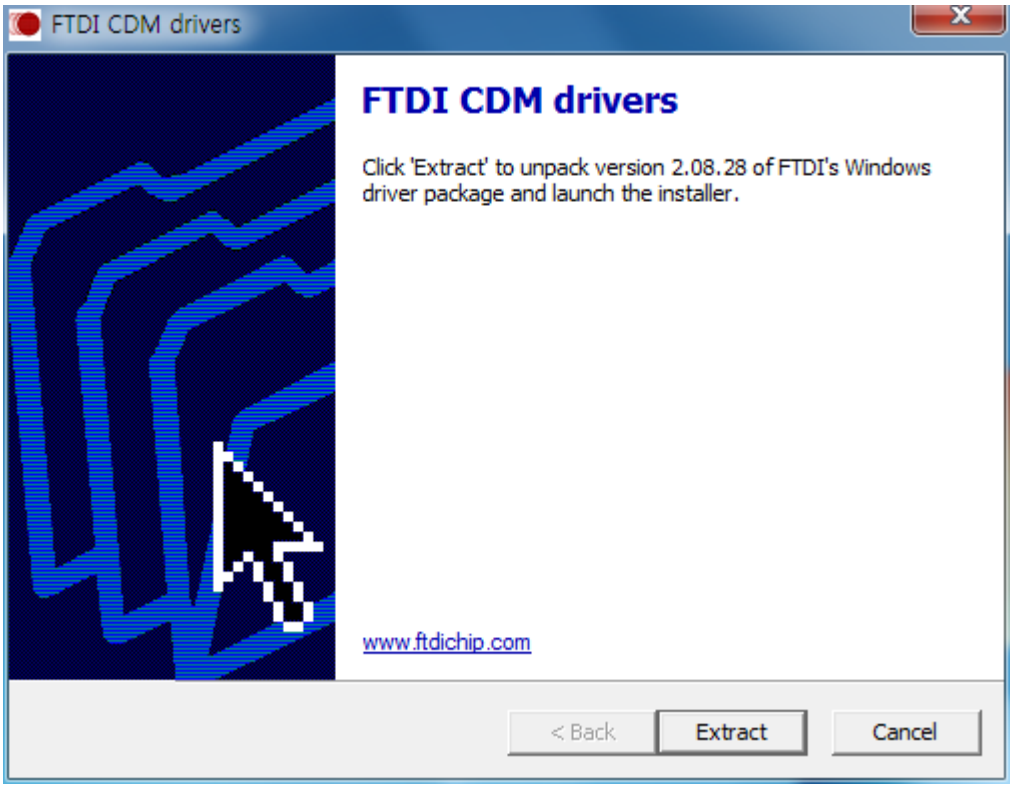


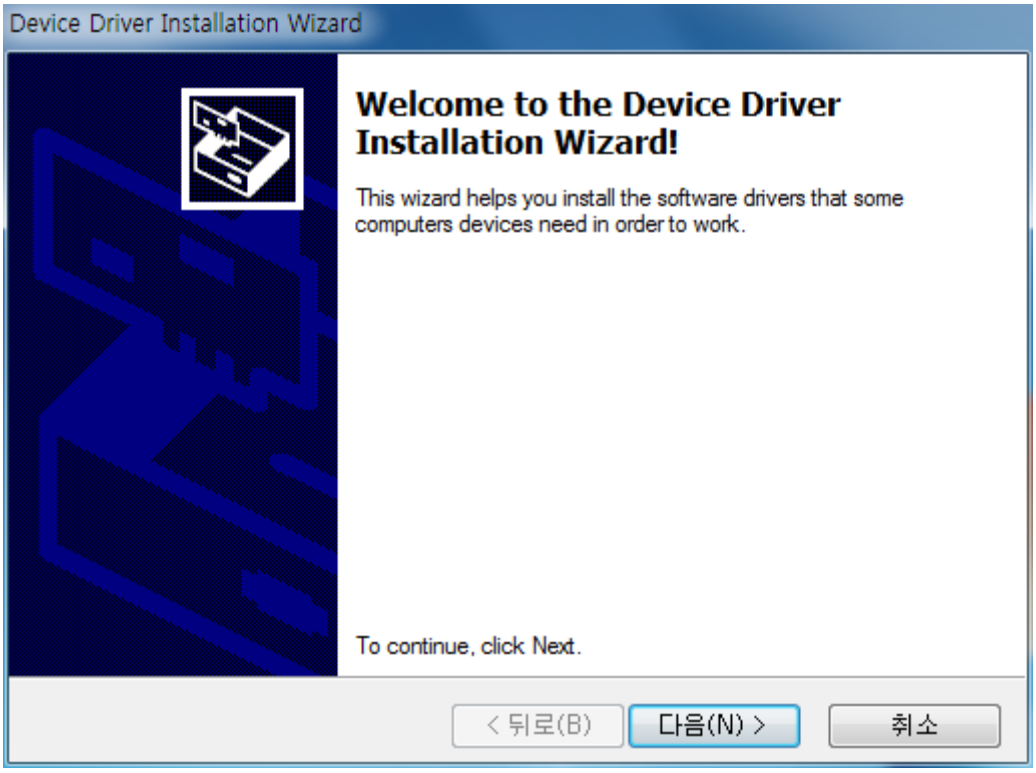
12.2 Windows 7 에서

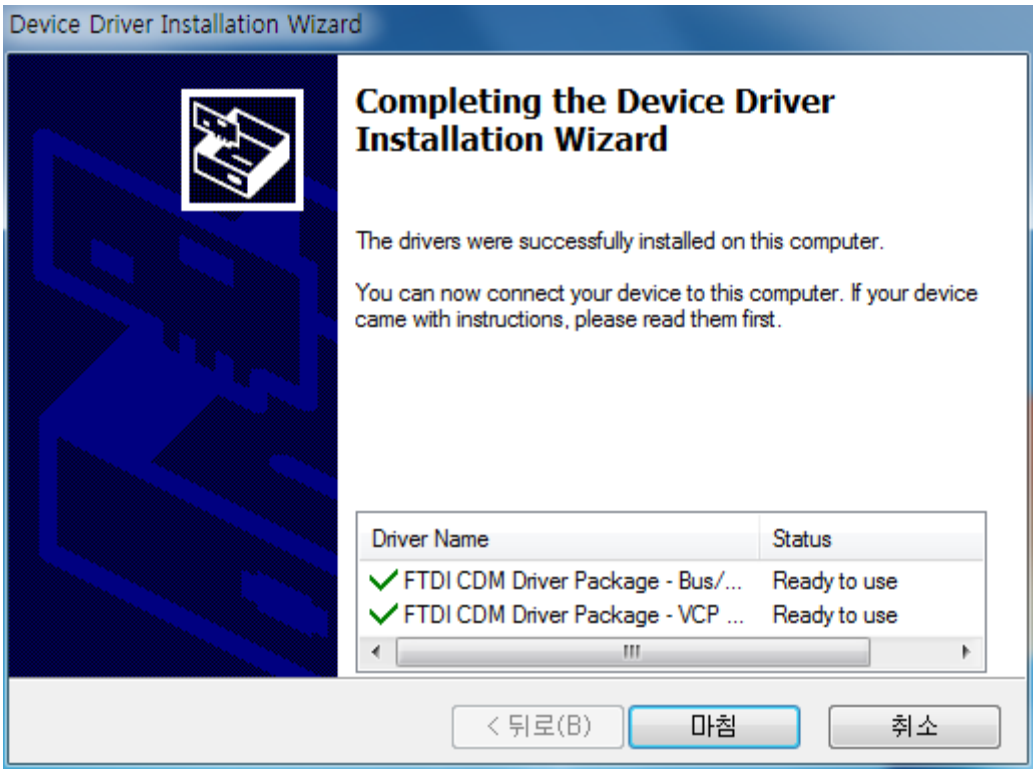
디바이스 드라이버를 설치하기에 앞서 먼저 easyDSP 용 USB POD 와 USB cable 및 제공된 Device Driver 가 있는지 확인하시길 바랍니다.

1) PC 에 easyDSP 를 연결하지 마시고, 제일 먼저 easyDSP 배포파일 중에 'Driver' 폴더 안에 있는 CDM212364_Setup.exe 를 실행합니다. 아래 그림과 같이 진행해 주세요.

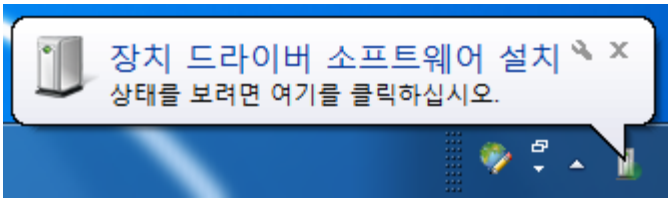

파일 열기 - 보안 경고
이 파일을 실행하시겠습니까?
이름: ...Users₩dwchung.LSPST₩Downloads₩CDM20828.exe
게시자: Future Technology Devices International Ltd
유형: 응용 프로그램
시작: C:₩Users₩dwchung.LSPST₩Downloads₩CDM20828....
실행(R)
취소
이 파일을 열기 전에 항상 확인(W)
이 형식의 파일은 사용자의 컴퓨터에 피해를 줄 수 있습니다. 신뢰할 수 있는 게시자로부터의 소프트웨어만 실행하십시오. 위험성


FTDI CDM drivers
FTDI CDM drivers
Click 'Extract' to unpack version 2.08.28 of FTDI's Windows driver package and launch the installer.
www.ftdichip.com
< Back
Extract
Cancel

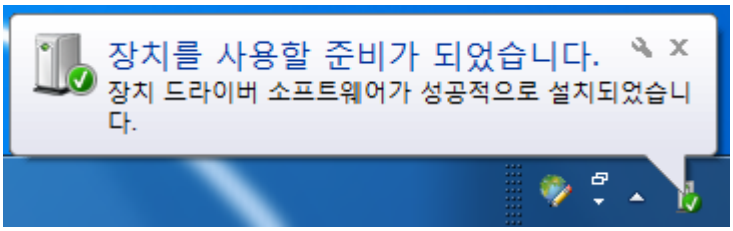




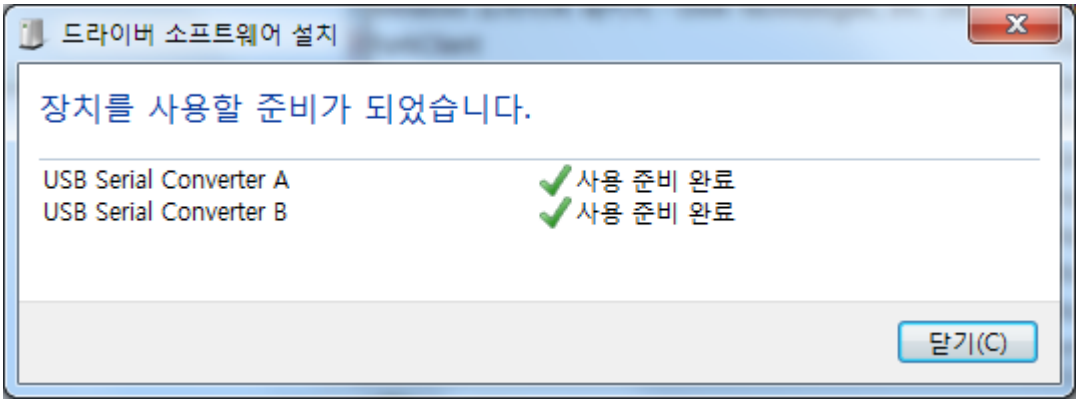
2) 이후 easyDSP 를 PC 에 연결하시면, 아래와 같이 트레이 창에 풍선 창이 나오게 됩니다.



약간 시간이 지나면, 풍선 창이 바뀌게 됩니다.



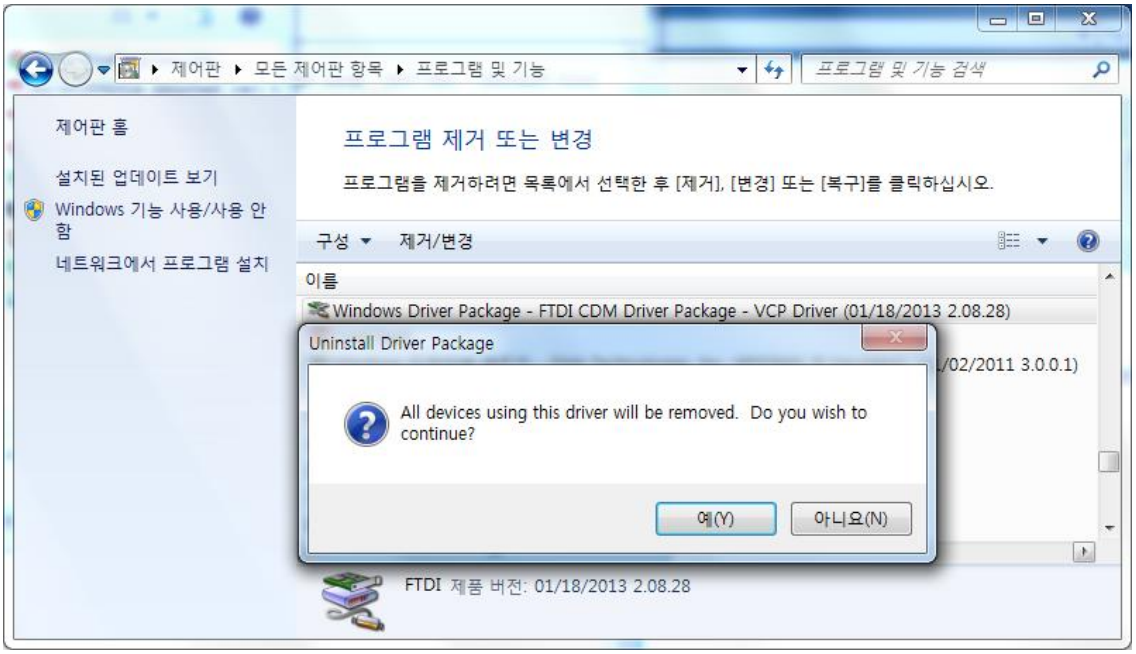
본 풍선 창을 클릭하면 아래와 같습니다.



이제 easyDSP 용 드라이버가 다 설치된 것입니다. 이후 사용 가능합니다.

3) 만약 상기 1)번의 과정을 거치지 않고 easyDSP 를 사용하면 오동작할 수 있습니다.

4) 드라이버를 제거하실 경우, 제어판 > 프로그램 > 프로그램 제거를 실행 한 후, 목록에서 "Windows Driver Package - FTDI CDM Driver Package - VCP Driver"를 선택하고 삭제해 주세요.



12.3 제거 방법

드라이버 제거

일반적인 윈도우즈 방식으로 드라이버를 제거가 어려울 경우, 하기 방식을 제안 드립니다.

https://www.ftdichip.com/Support/Utilities.htm#CDMUninstaller 를 방문하셔서
https://www.ftdichip.com/Support/Utilities/CDMUninstaller_v1.4.zip 프로그램을 다운로드 받은 후,
https://www.ftdichip.com/Support/Utilities/CDM_Uninst_GUI_Readme.html 가이드라인에 따라 제거 바랍니다.

즉, 먼저 Vendor ID, Product ID 를 Add 한 후, Remove Devices 실행합니다.