

HOT_XE164_ASC-Link

Never stop thinking

Requirements

IDE
[Compiler, Assembler,
Linker Locator, Debugger]

TASKING

DAS
any tool
any wire
any device

Auto Code
Generation Tool



Windows Operating System
PC with USB Interface



Target System

UConnect



Easy Kit XE164



Goal

- HOT_XE164_ASC-Link is a Hands-On Training material created for the Infineon Real Time Signal Controller XE164
- The training's main aim is to become familiar with the software toolchain and to implement an ASC interconnection
- Two ASC interfaces are set to transmit and to receive characters from each other
- One ASC interface is set to transmit and to receive characters from a terminal program

Note:
This Hands-On Training is based on:

- DAvE 2.1 Mothersystem
- DAvE .dip file version 2.0
- Tasking Classic v8.7r2

Table of contents

Introduction

Hardware setup

Use DAvE to create HOT_XE164_ASC-Link

Use Tasking EDE to compile and debug your program

Basic Principles

- HOT_XE164_ASC-Link shows the user from the scratch how to do the hardware setup for an ASC link and how to generate software for the Infineon XE164 with DAvE and the compiler tool chain in the following steps:

1. First the user do the hardware interconnection for an ASC link by two wires to get an bidirectional ASC interconnection

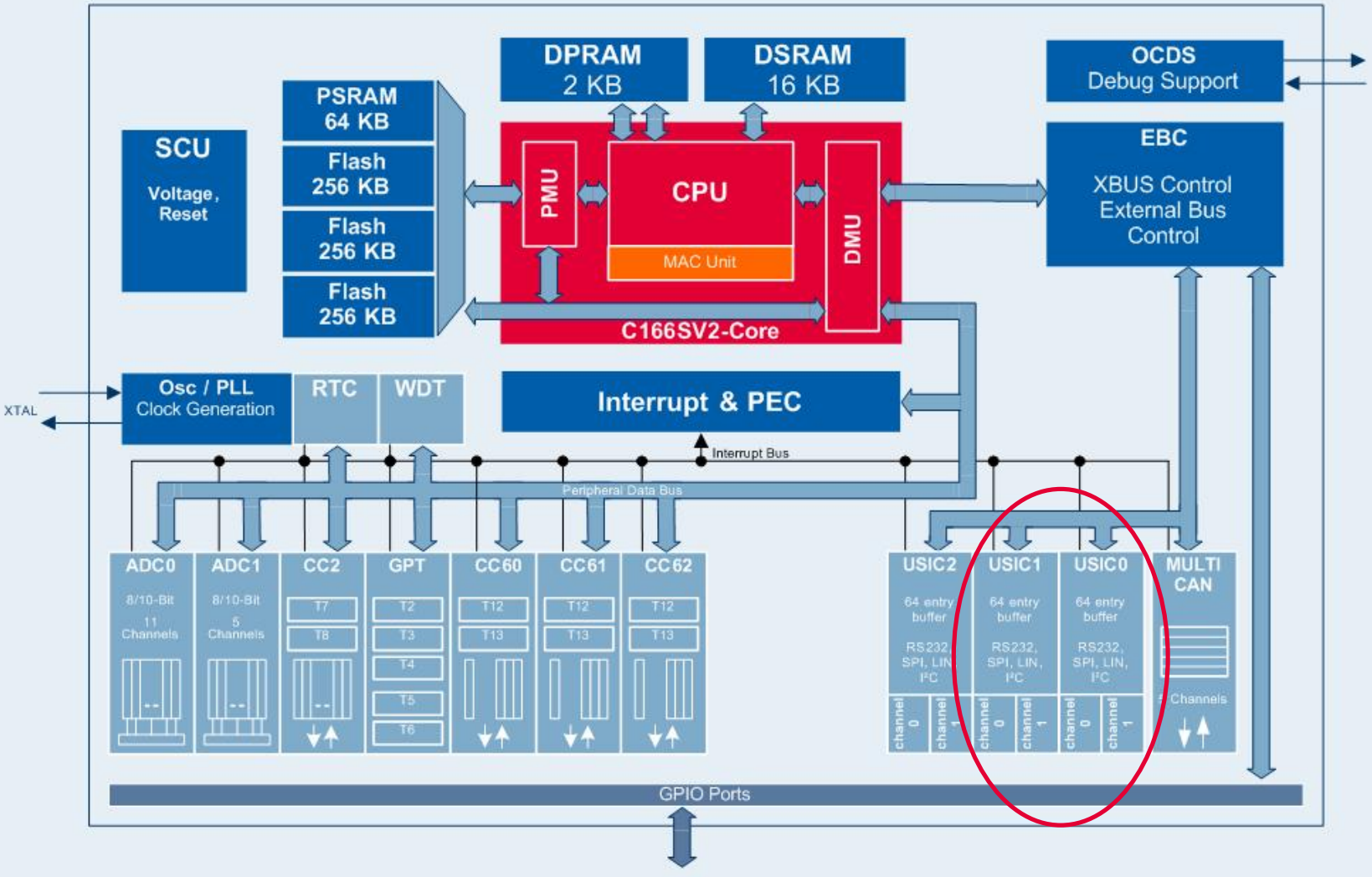
2. Referring to the hardware setup the user creates a new project in DAvE and configures the device by following the detailed instructions.

3. After code generation by DAvE, the user
   - switches to Tasking EDE and opens the created project
   - this include the C files created by DAvE and the assembler startup file
   - adds some user code for certain functionality
   - compiles, assembles, links and locates the project

Basic Principles

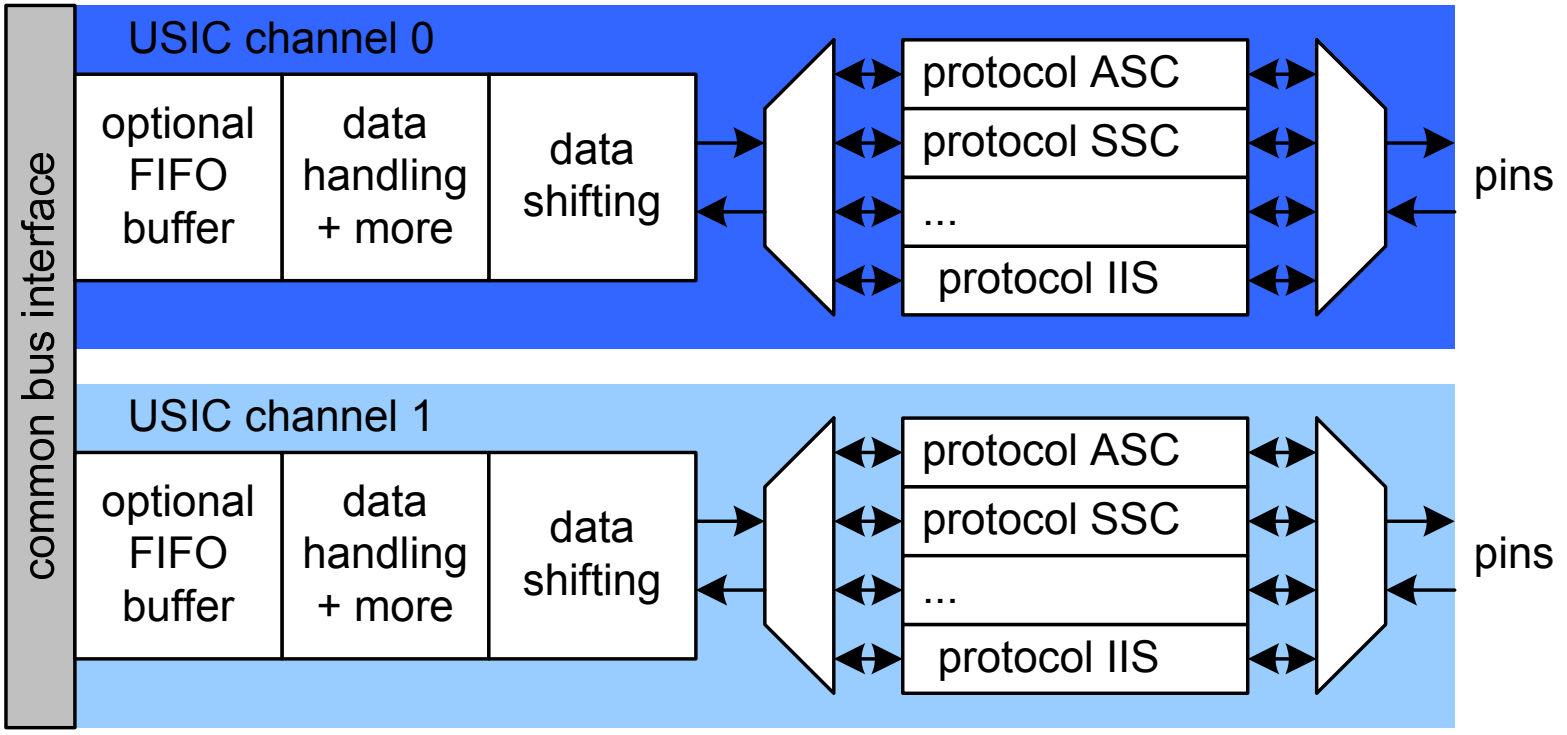
4. After compilation with EDE, the user
   - takes the included debugger to download and debug the program
   - opens a Terminal Window (e.g. HyperTerminal or MTTTY) with the configured Baud rate and type in some characters
   - all characters typed in from the keyboard will be displayed twice on the terminal program

Introduction to the new Universal Serial Interface Channel (USIC)



Introduction to the new Universal Serial Interface Channel (USIC)

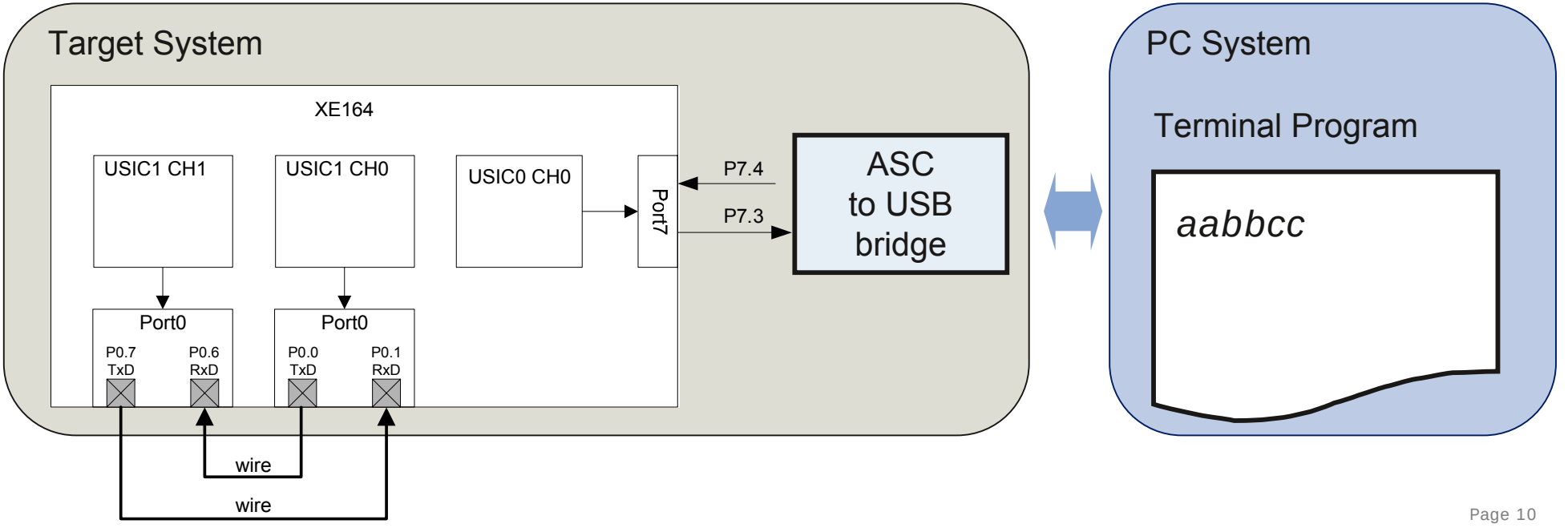
The USIC module combines the ability to receive and to transmit data frames in each of the following protocols in full duplex mode.



- **Each USIC channel**
  - is capable of handling ASC, SSC, LIN, IIC and IIS
  - can be individually configured (incl. baud rate generation)
  - can handle full duplex data transfers
  - can be connected to several pins (incl. alternative locations)
  - can be reprogrammed without chip reset
  - contains a flexible interrupt generation part (interrupt compressing and routing)

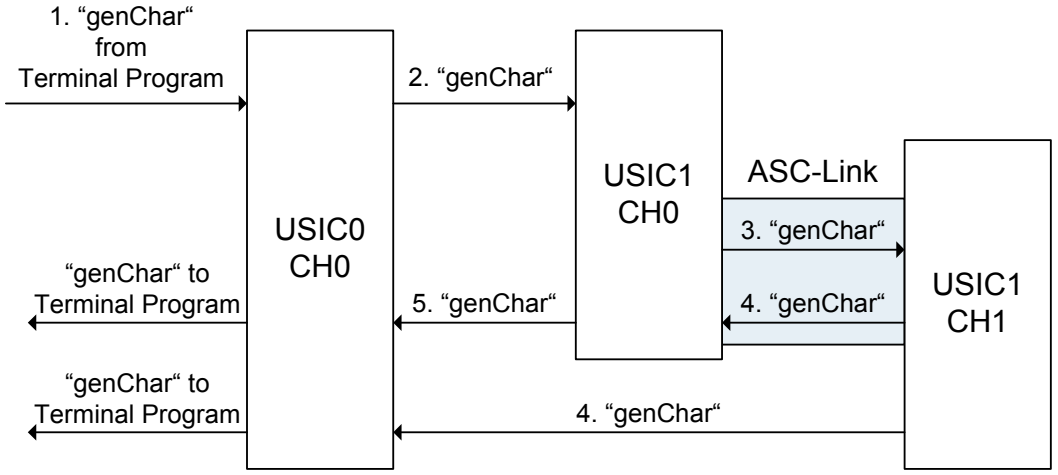
Hardware Setup

- Three Universal Serial Interface Channels (USIC) are configured as ASC interfaces
- Channel 0 of the USIC module 0 is linked to an ASC to USB bridge on the target system – characters can be received and transmitted to a terminal program
- Channel 0 and channel1 of USIC1 are linked together as full duplex ASC interconnection



Software Description (1)

- The data flow of the example software is the following:
  1. USIC0 CH0 receives a generic character (“genChar”) from the Terminal Program
  2. USIC0 CH0 passes genChar to USIC1 CH0
  3. The latter sends the character to USIC1 CH1 by the established ASC link



Software Description (2)

4. Once the character is received by USIC1 CH1, first it sends the character to USIC0 CH0 (and than to the terminal program) to indicate correct transmission from USIC1 CH0 to USIC1 CH1. Second, it sends the received character back to USIC1 CH0 by the established ASC link
5. Finally USIC1 CH0 passes the character received from USIC1 CH1 to USIC0 CH0 (and than to the terminal program) to indicate correct transmission from USIC1 CH1 to USIC1 CH0.

- A correct ASC-Link data handling is indicated by double display of the keyboard input

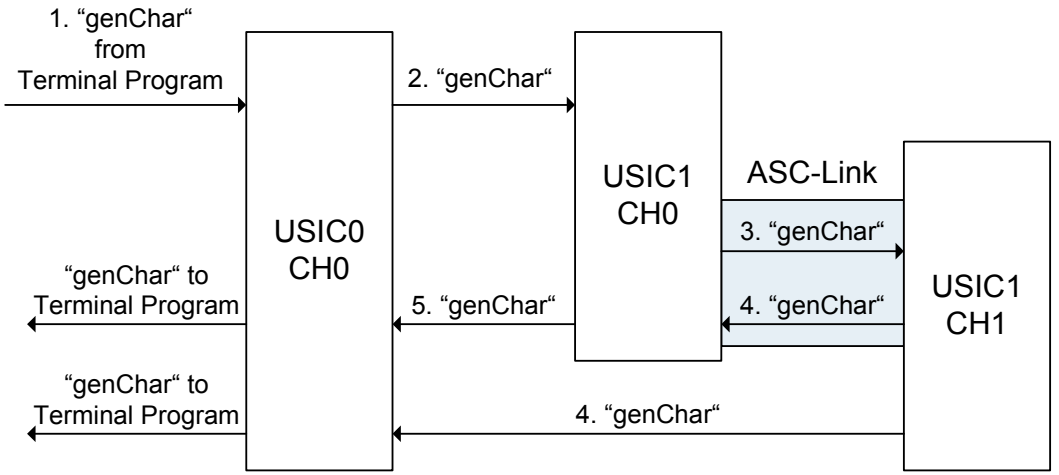


Table of contents

Introduction

Hardware setup

Use DAvE to create HOT_XE164_ASC-Link

Use Tasking EDE to compile and debug your program

Plug in the wires on the target system

- Take two wires to connect channel 0 and channel1 of USIC1
- Associated pads are in the following positions:
  - UConnect



UCONNECT (front view)

15	13	11	9	7	5	3	1
CANH	P0.4	P0.5	P0.3	P0.2	P5.9	P15.0	GND
CANL	P0.7	P0.6	P0.0	P0.1	P5.8	P5.0	+5V
16	14	12	10	8	6	4	2

Plug in the wires on the target system

- Easy Kit XE164



Plug X107

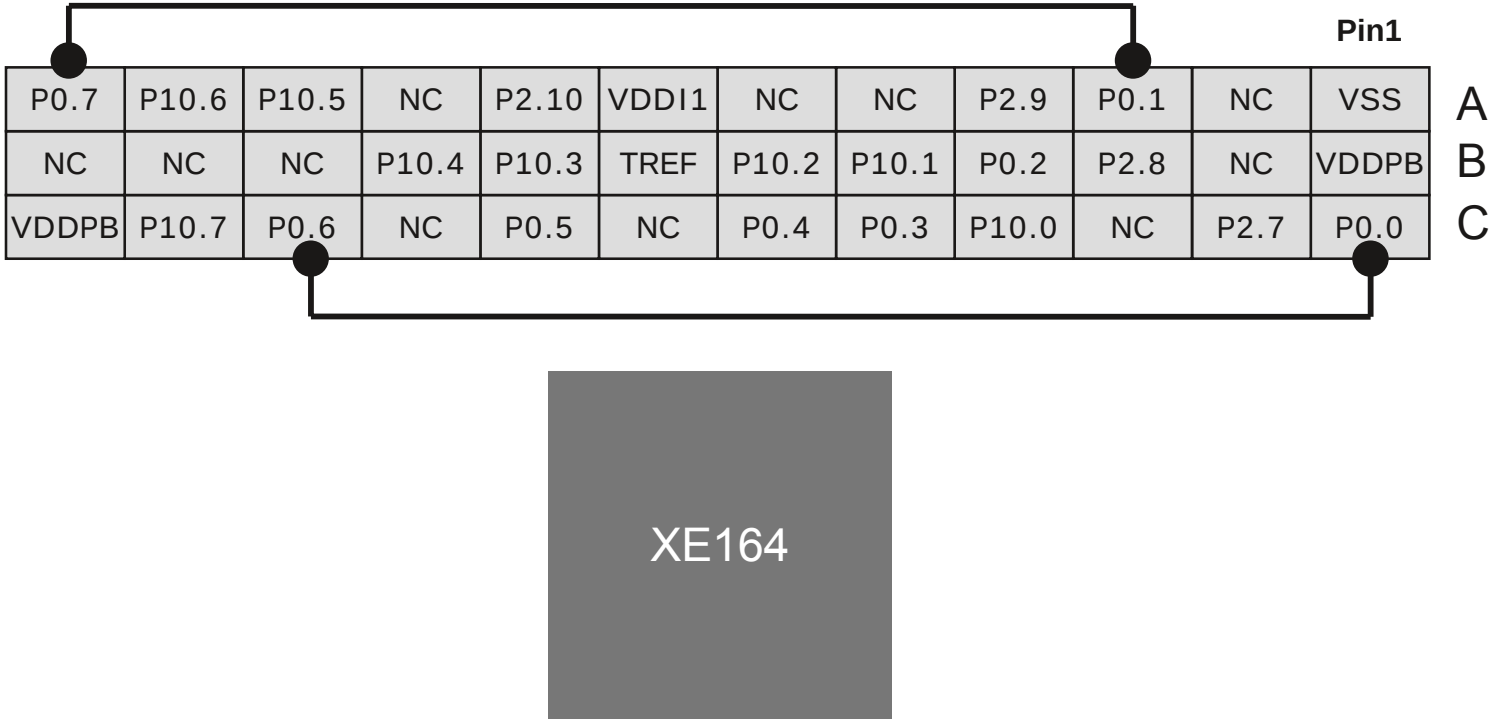


Table of contents

Introduction

Hardware setup

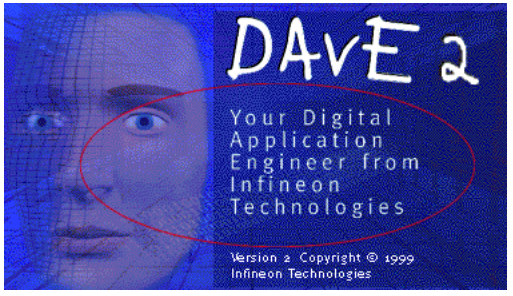
Use DAvE to create HOT_XE164_ASC-Link

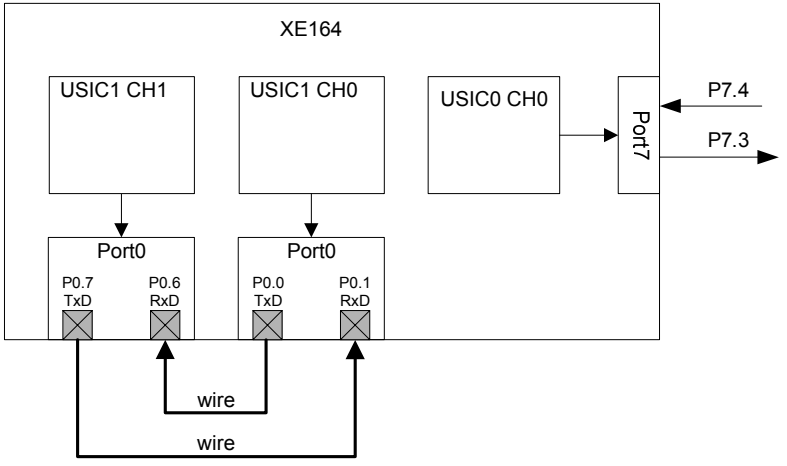
Use Tasking EDE to compile and debug your program

Use DAvE to create your code

Use DAvE to ease-up the USIC setup, it generates all the necessary code.
Dave creates all the initialization files needed for each peripheral you want to use.
This is a real **advantage** and **saves** a lot of **time**.

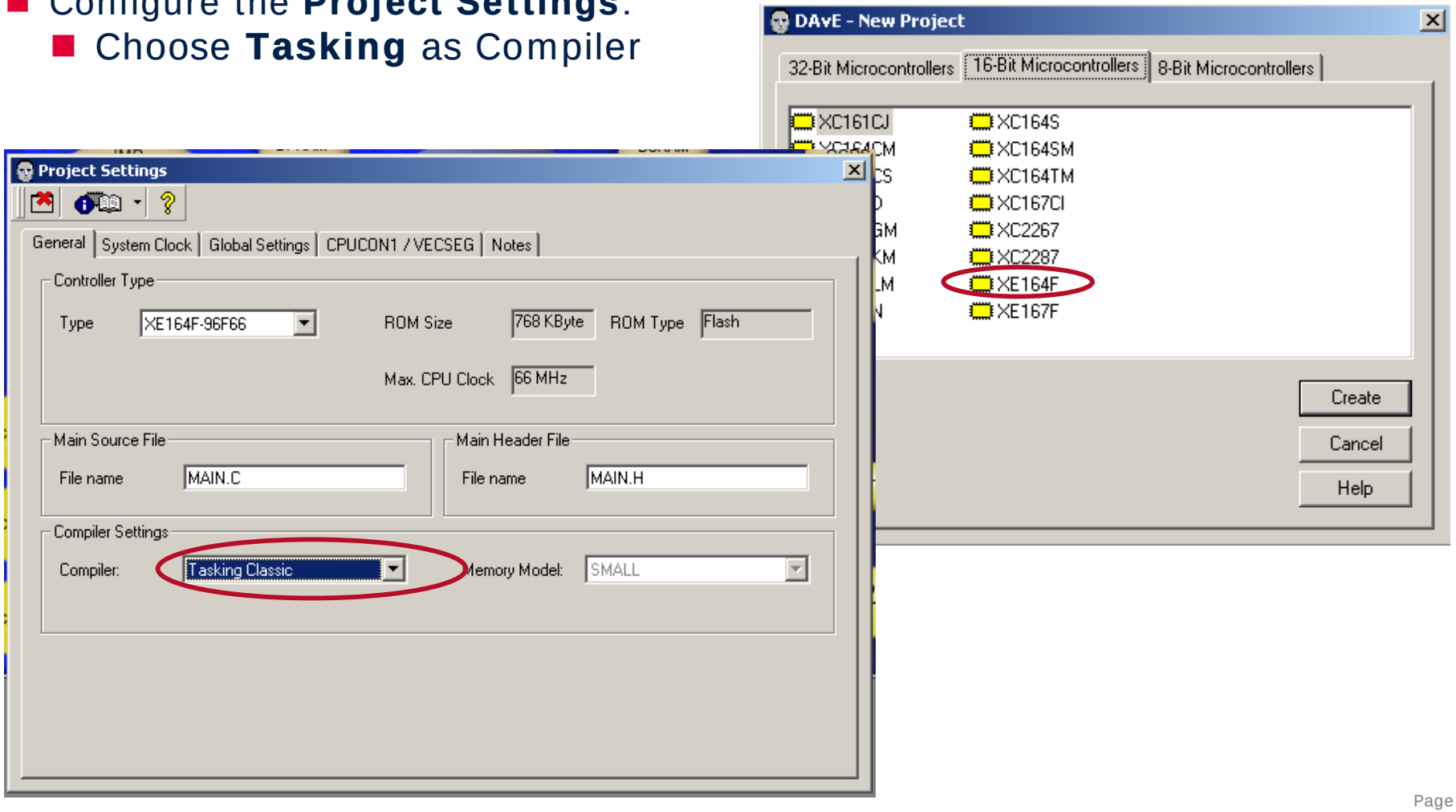
Launch **DAvE**





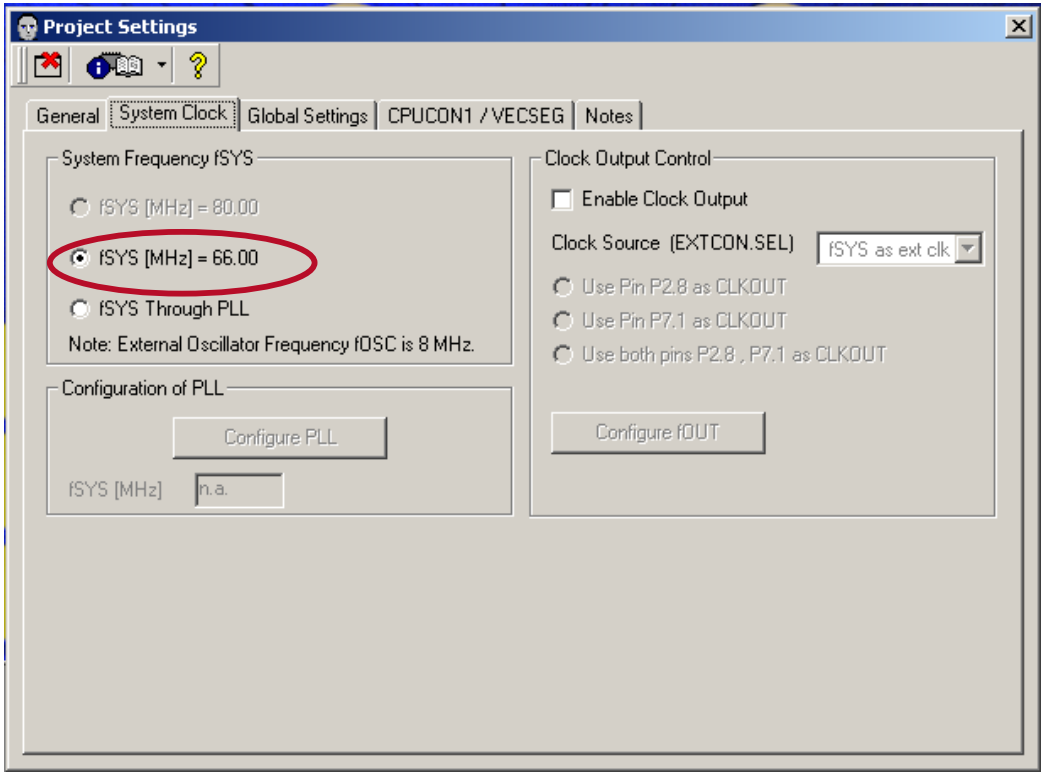
Use DAvE to create your code

- **Create** a **new** Project: **File → New**
  Choose the **XE164F** in the **16 bits** microcontroller tab
- Configure the **Project Settings**:
  - Choose **Tasking** as Compiler



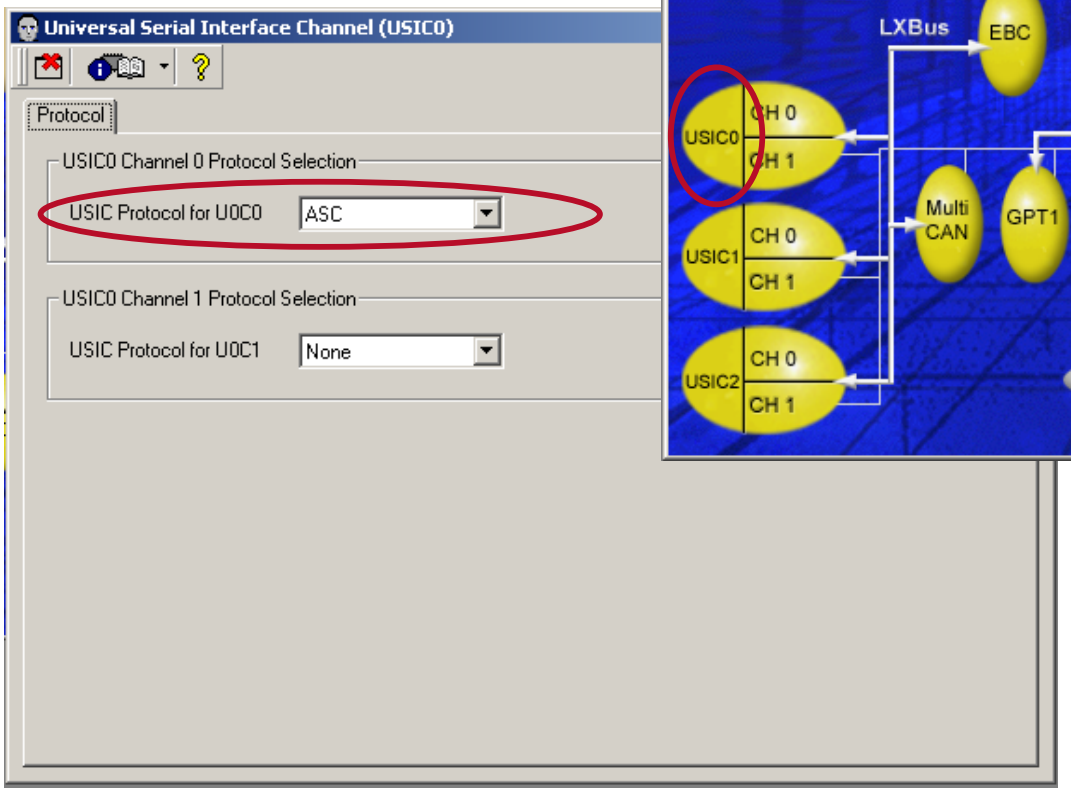
Use DAvE to create your code

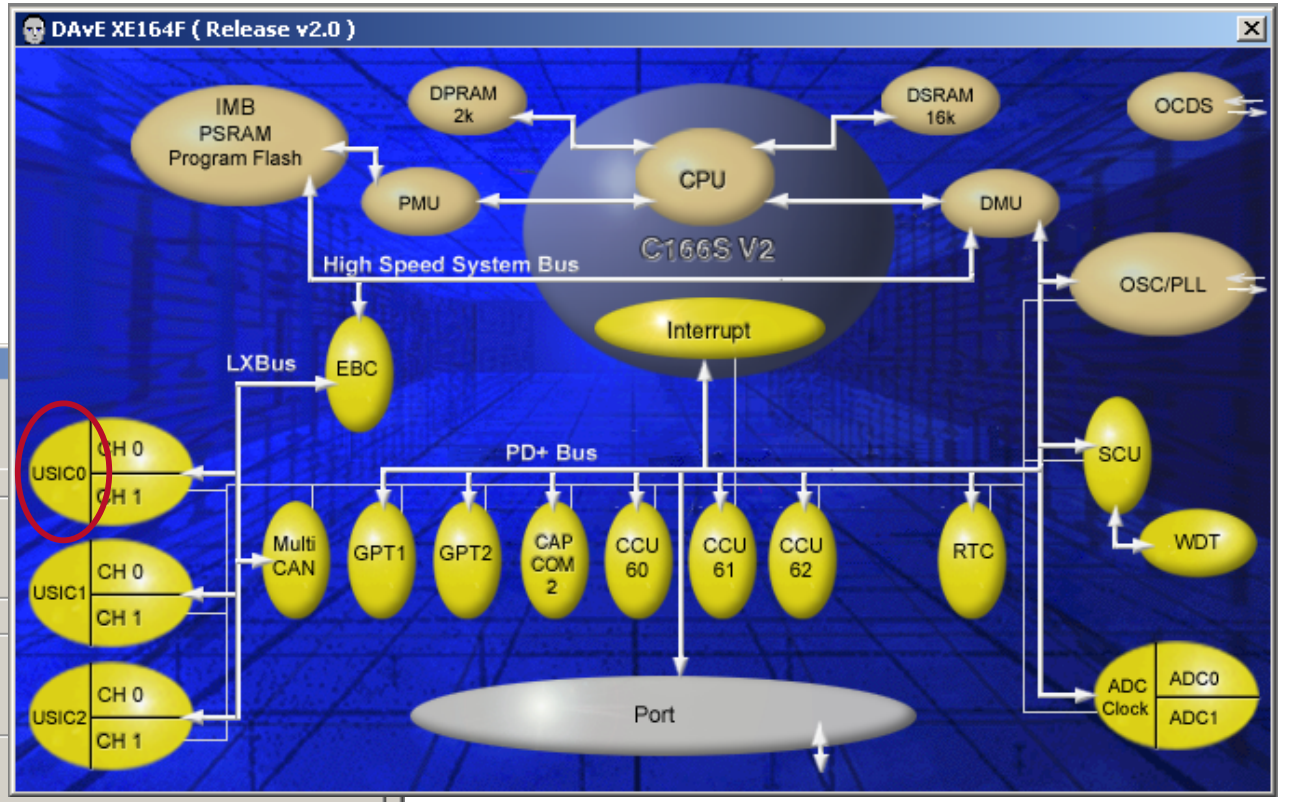
- The System Frequency is set per default to 66MHz with 8MHz input clock on XTAL1 pin



Use DAvE to create your code

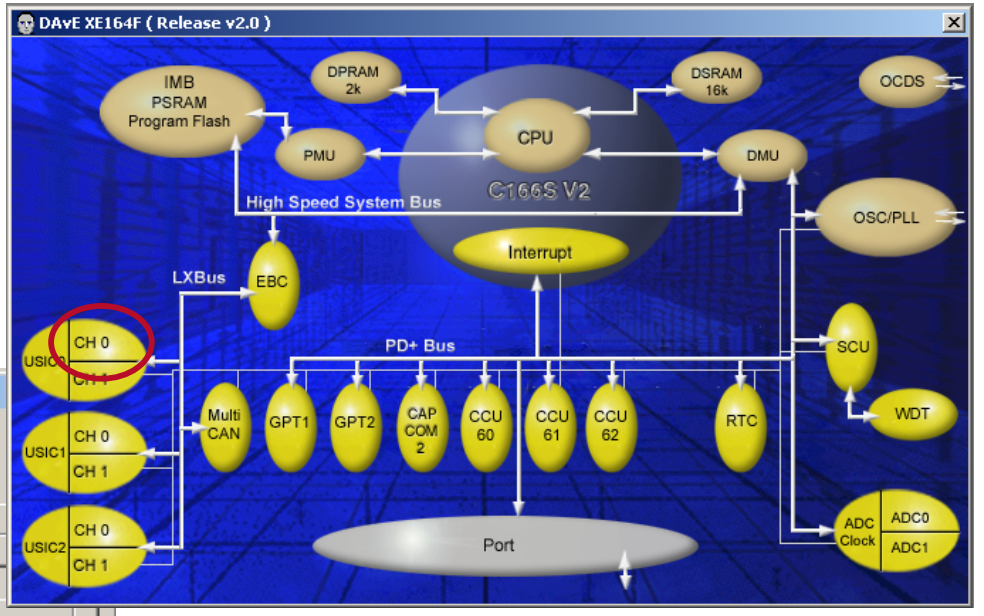
- Click on USIC0
- Enable the ASC Protocol for USIC0 channel 0

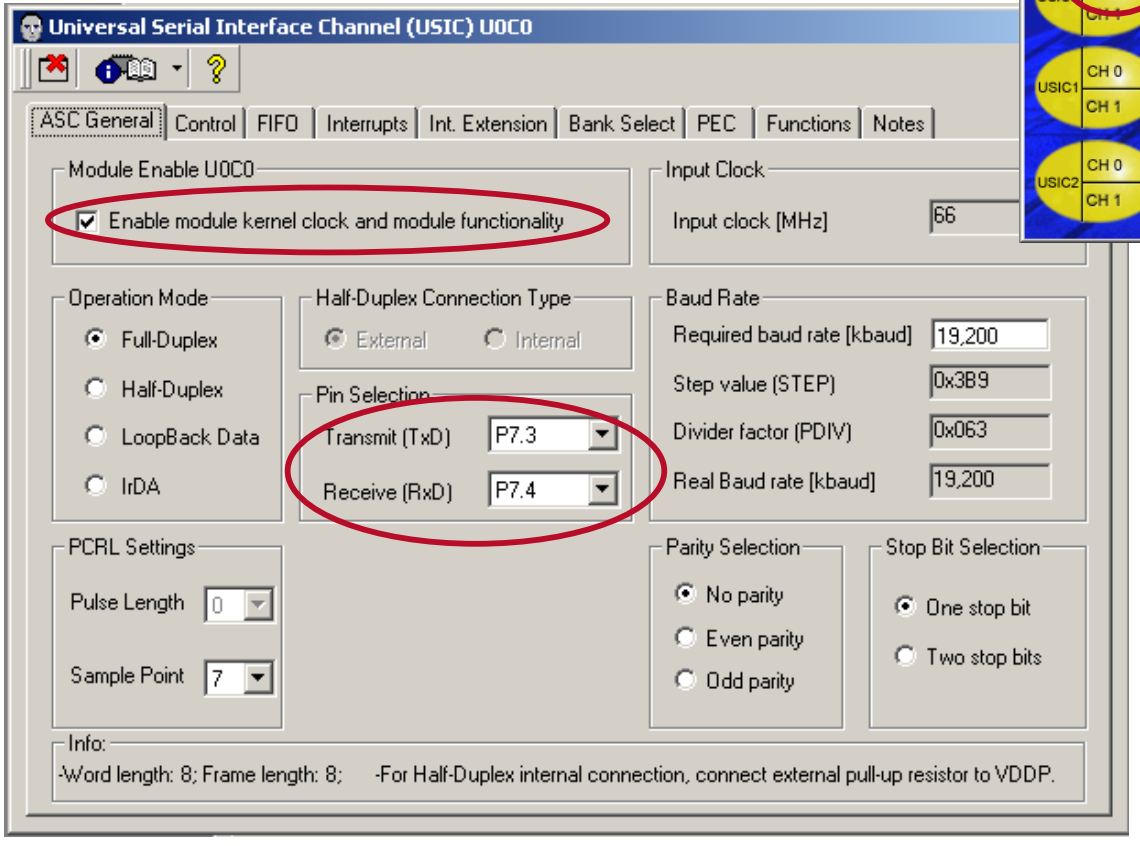




Use DAvE to create your code

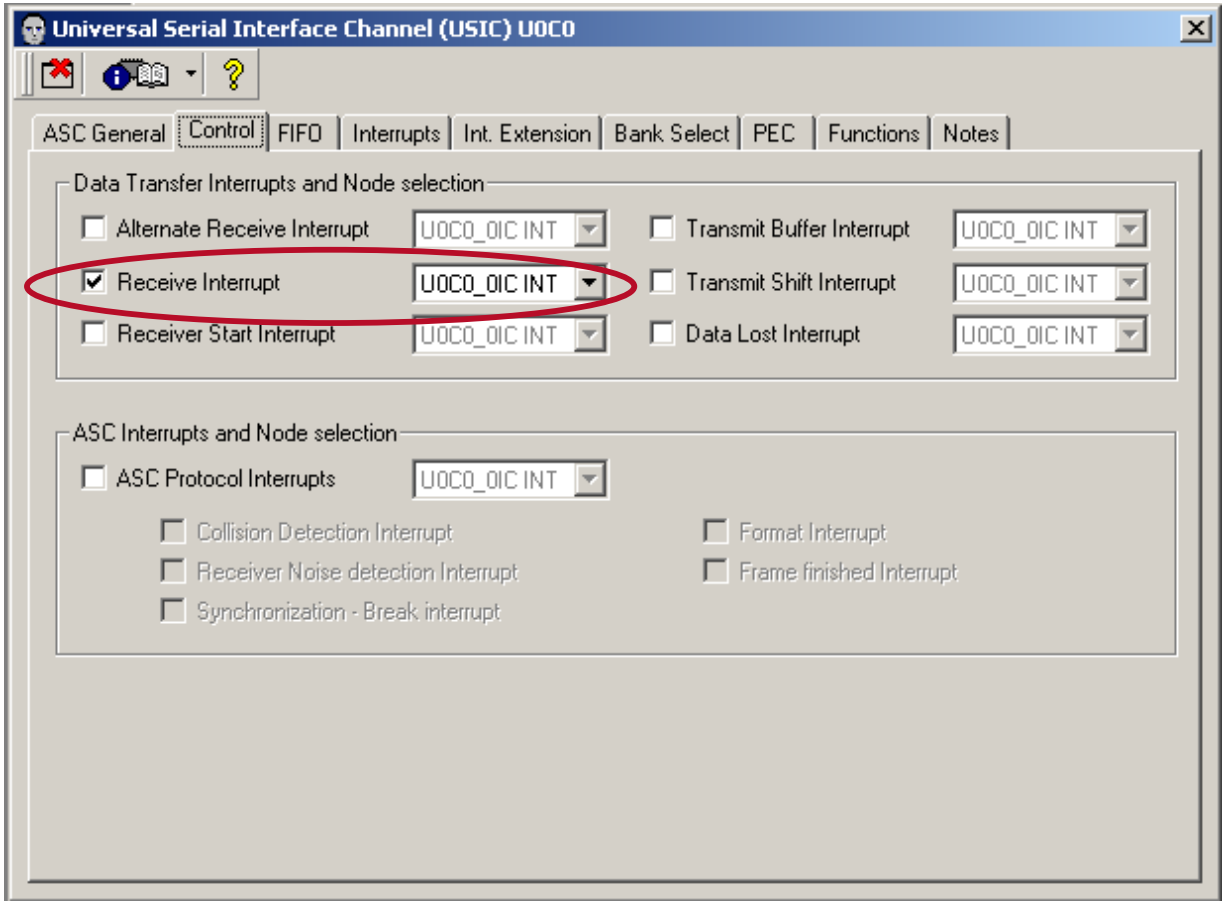
- Click on CH 0 of the USIC 0
- Enable the module
- Select the Pins for TxD and RxD
- Baudrate is 19,2 kBaud
- No parity Bit and 1 Stop Bit is used for the ASC protocol in this example





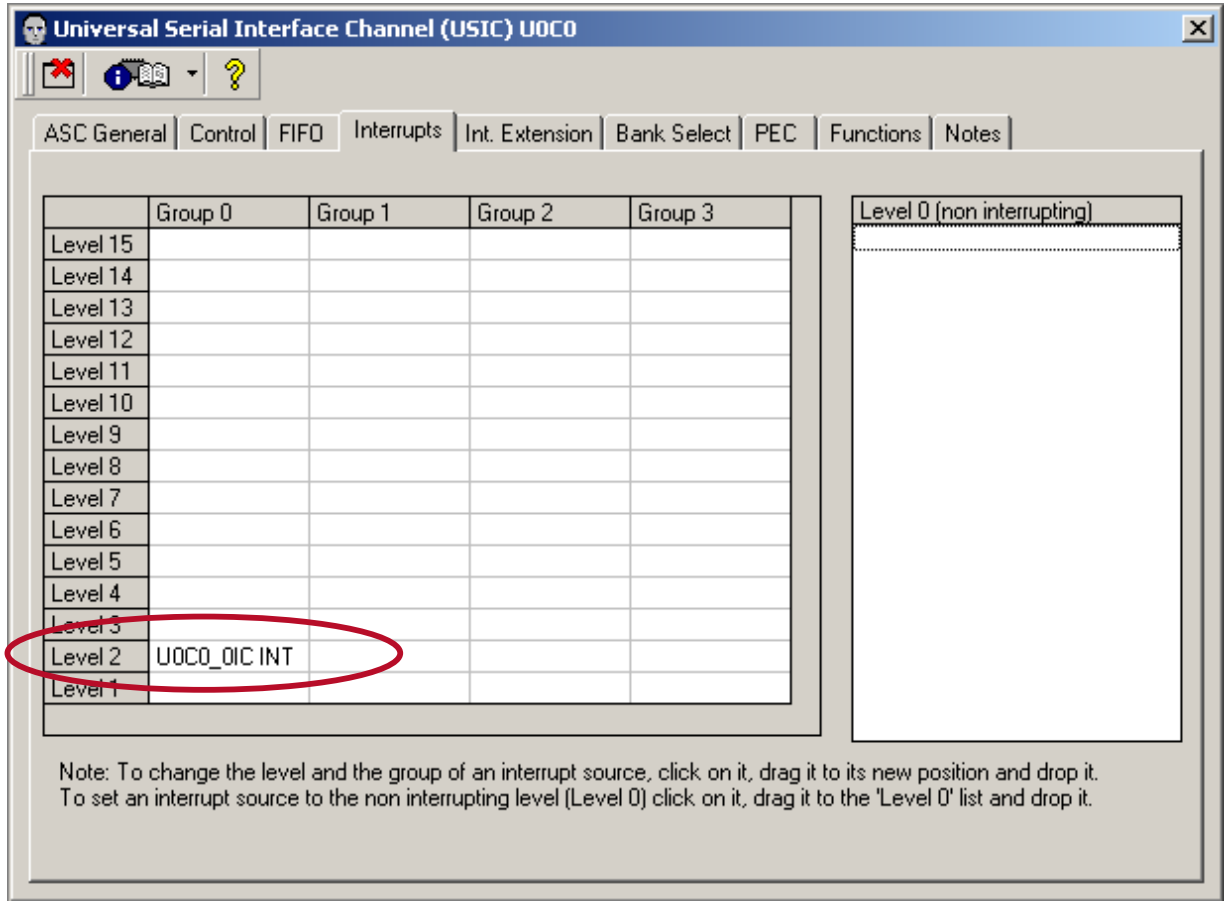
Use DAvE to create your code

- Click on Control of the USIC 0
- Enable Receive Interrupt



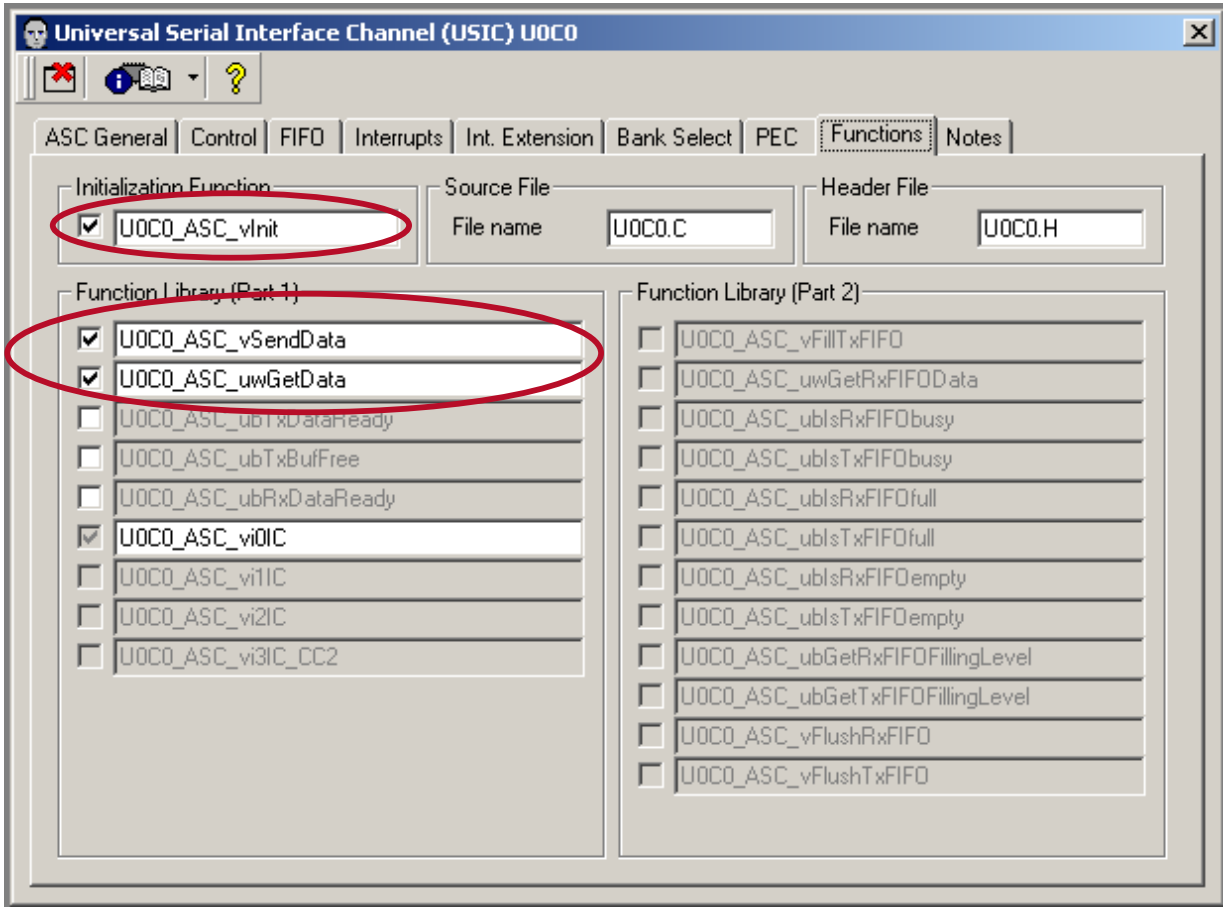
Use DAvE to create your code

- Select Interrupt Level 2 for the USIC 0 channel 0 interrupt by drag&drop



Use DAvE to create your code

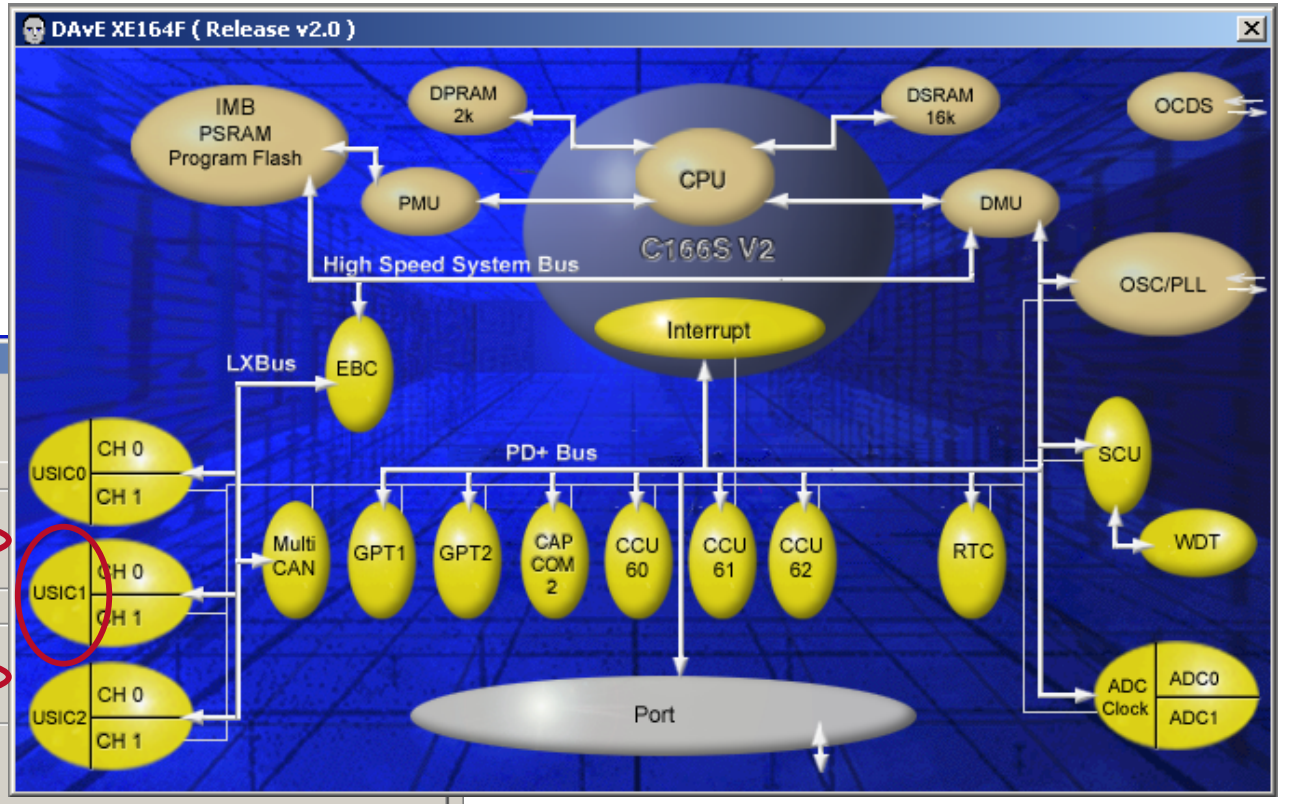
- Enable the USIC 0 Ch 0 ASC init function
- Select additional functions
  - U0C0_ASC_vSendData
  - U0C0_ASC_uwGetData



Use DAvE to create your code

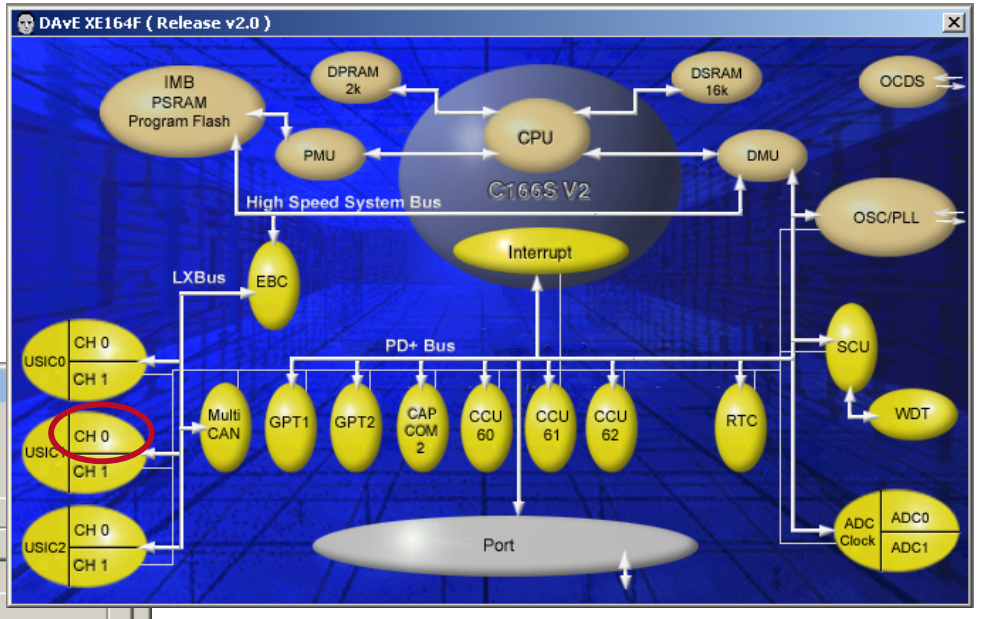
- Click on USIC1
- Enable the ASC Protocol for USIC1 channel 0 and channel1

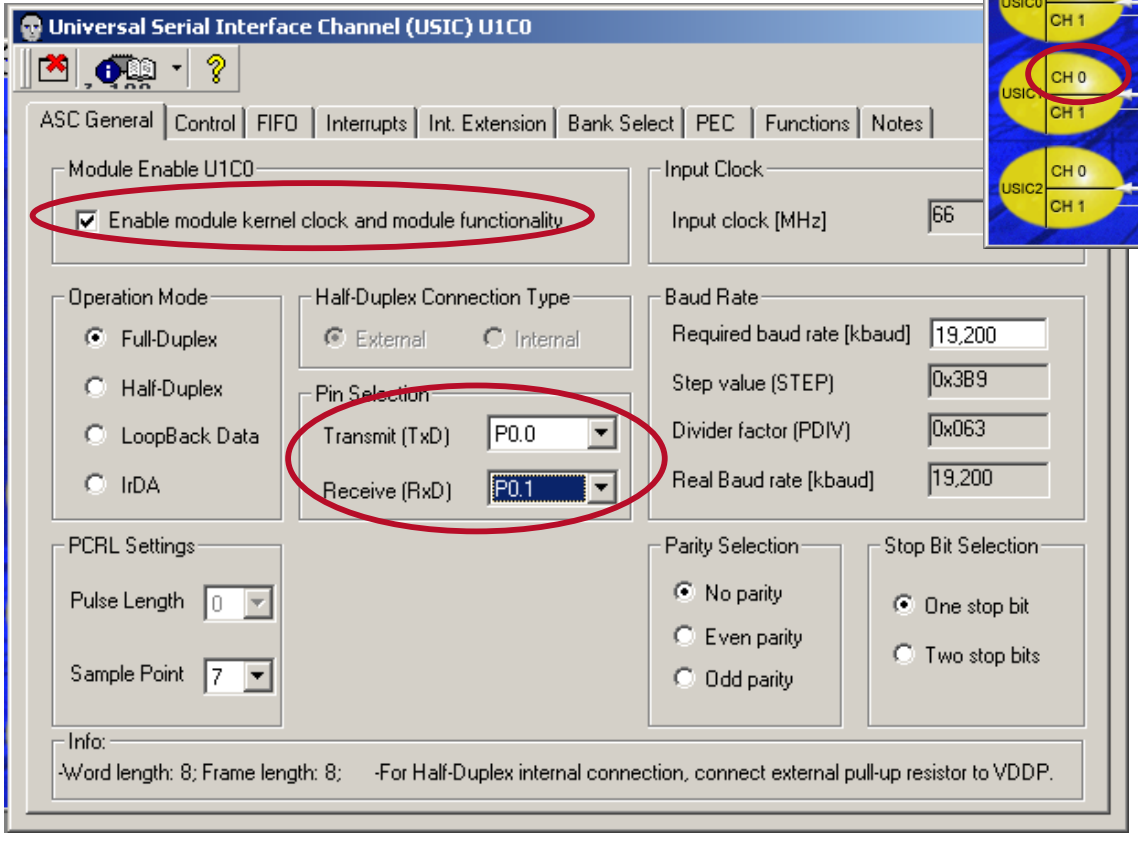




Use DAvE to create your code

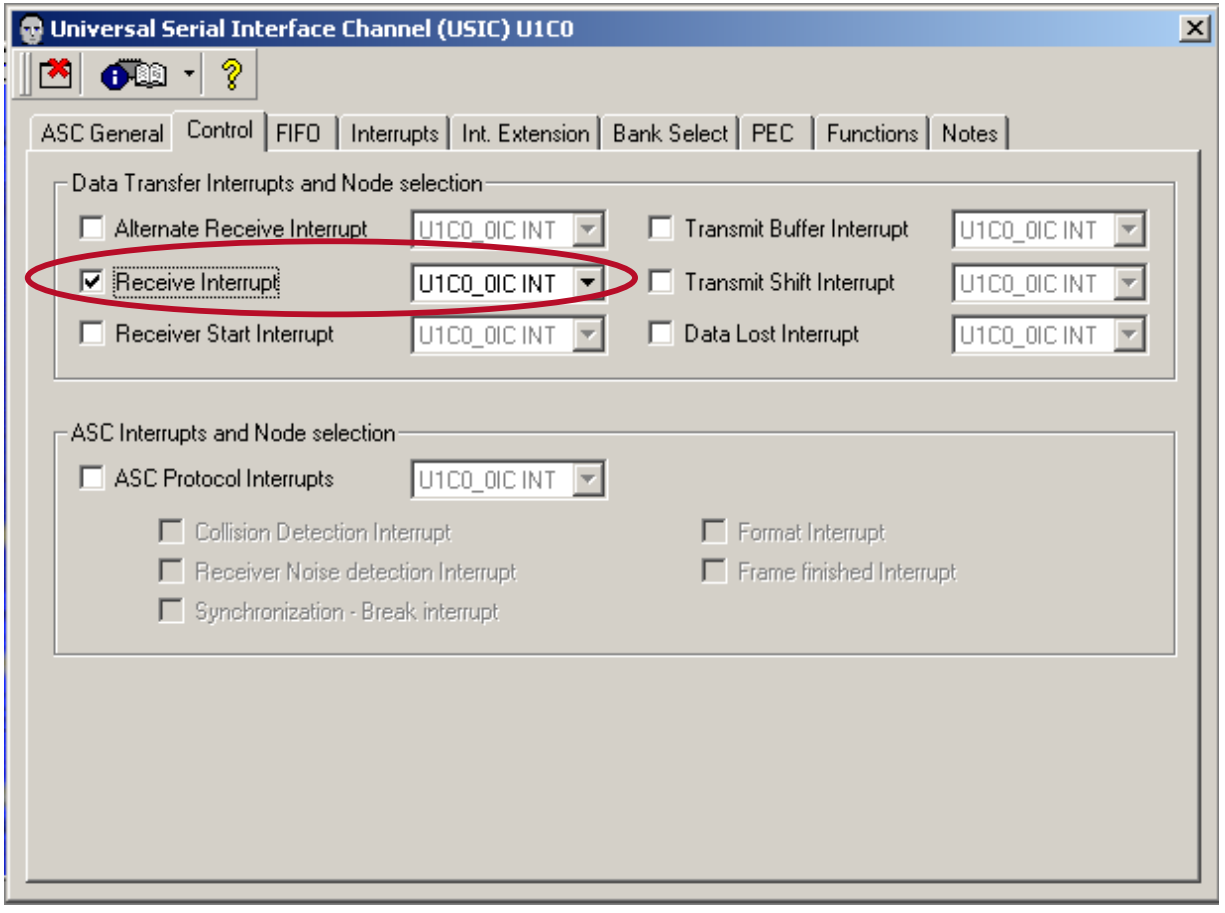
- Click on CH 0 of the USIC 1
- Enable the module
- Select the Pins for TxD and RxD
- Baudrate is 19,2 kBaud
- No parity Bit and 1 Stop Bit is used for the ASC protocol in this example





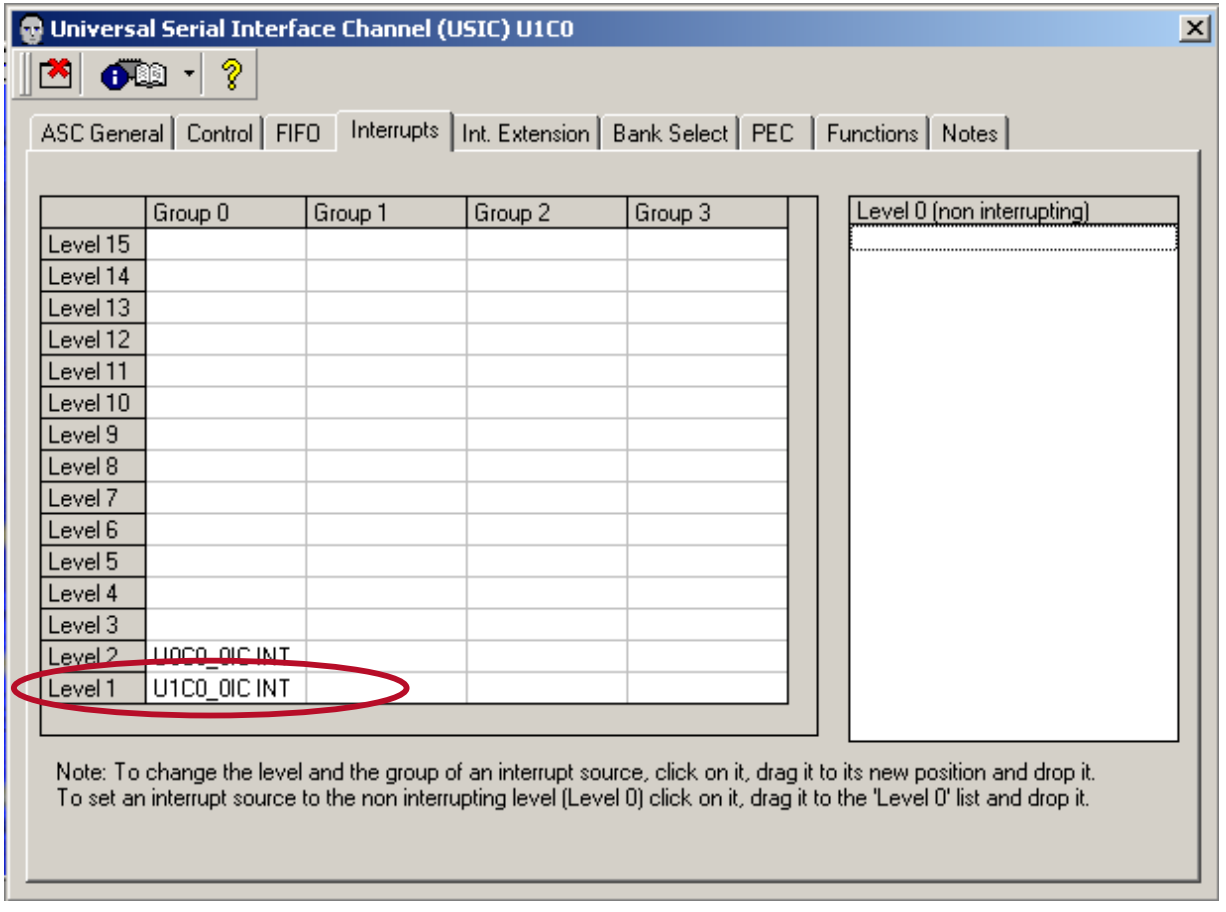
Use DAvE to create your code

- Click on Control of channel 0
- Enable Receive Interrupt



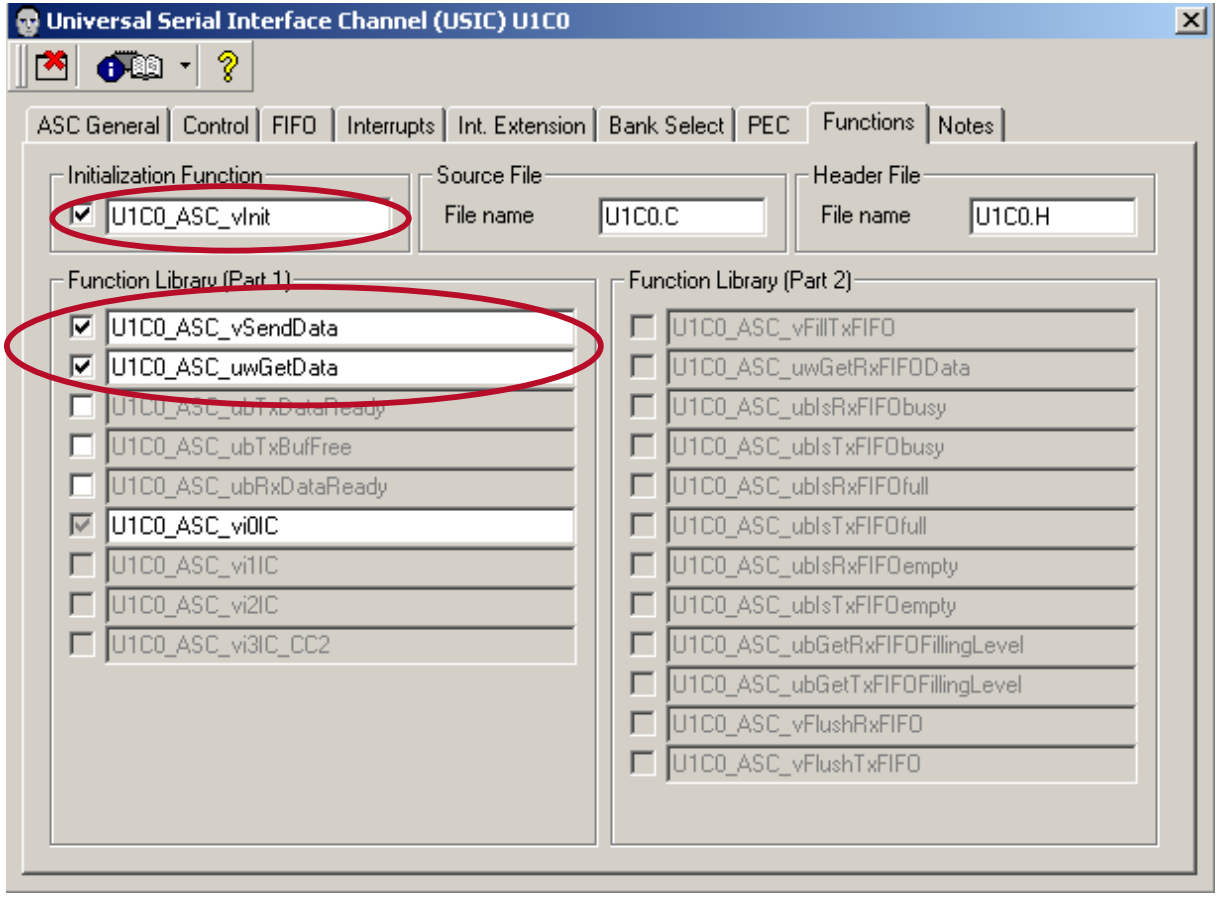
Use DAvE to create your code

- Select Interrupt Level 1 for the USIC 1 channel 0 interrupt by drag&drop



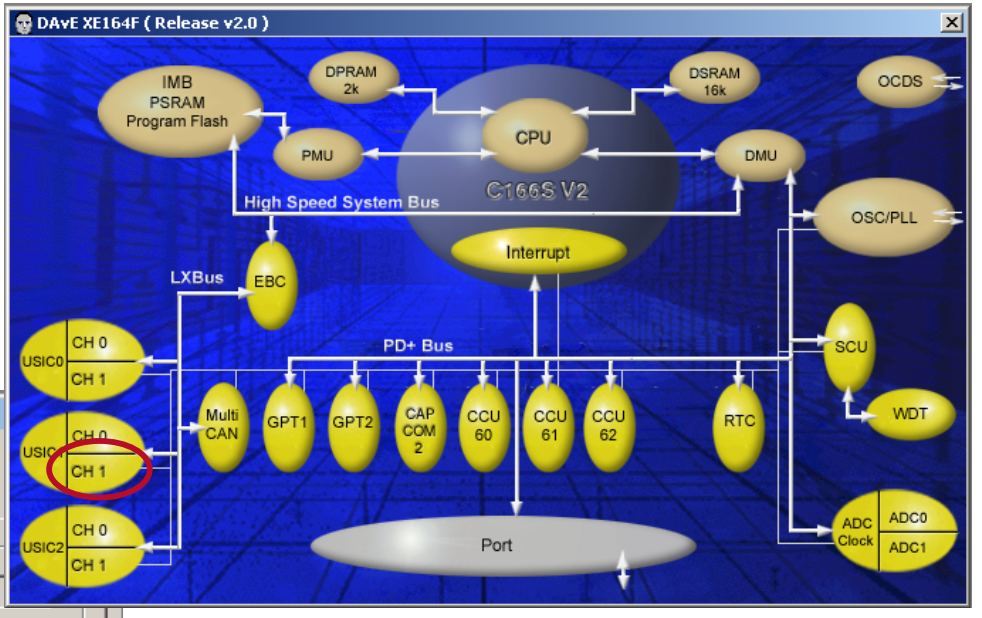
Use DAvE to create your code

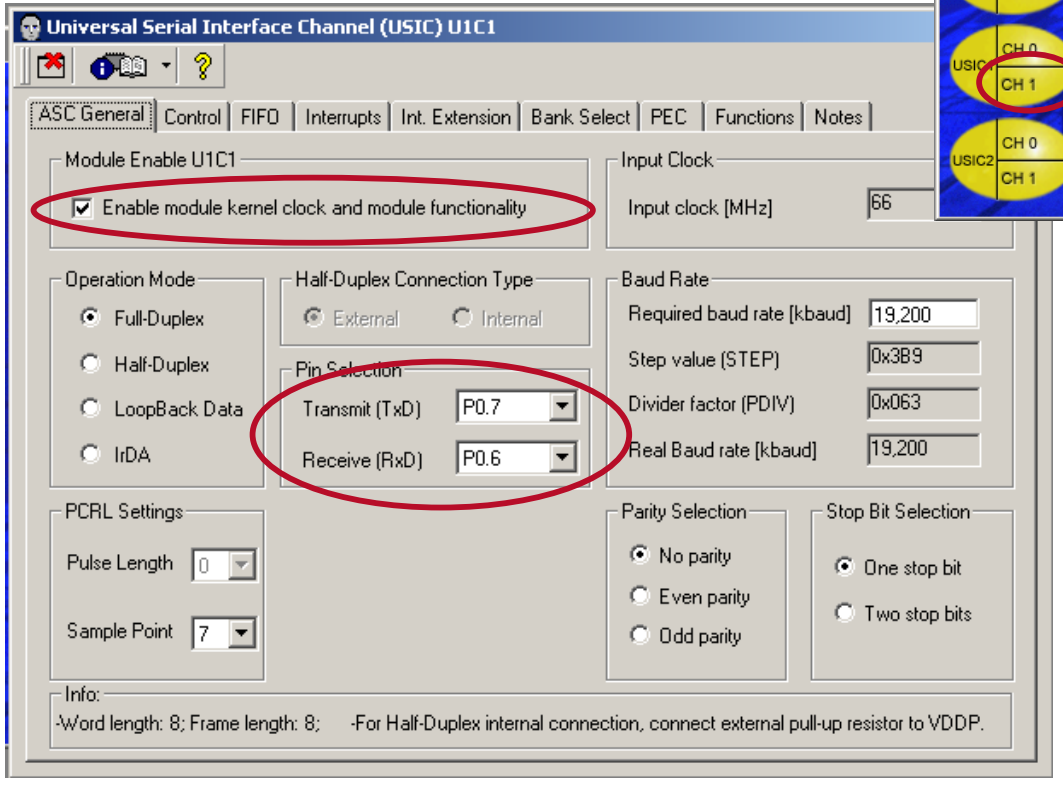
- Enable the USIC 1 Ch 0 ASC init function
- Select additional functions
  - U1C0_ASC_vSendData
  - U1C0_ASC_uwGetData



Use DAvE to create your code

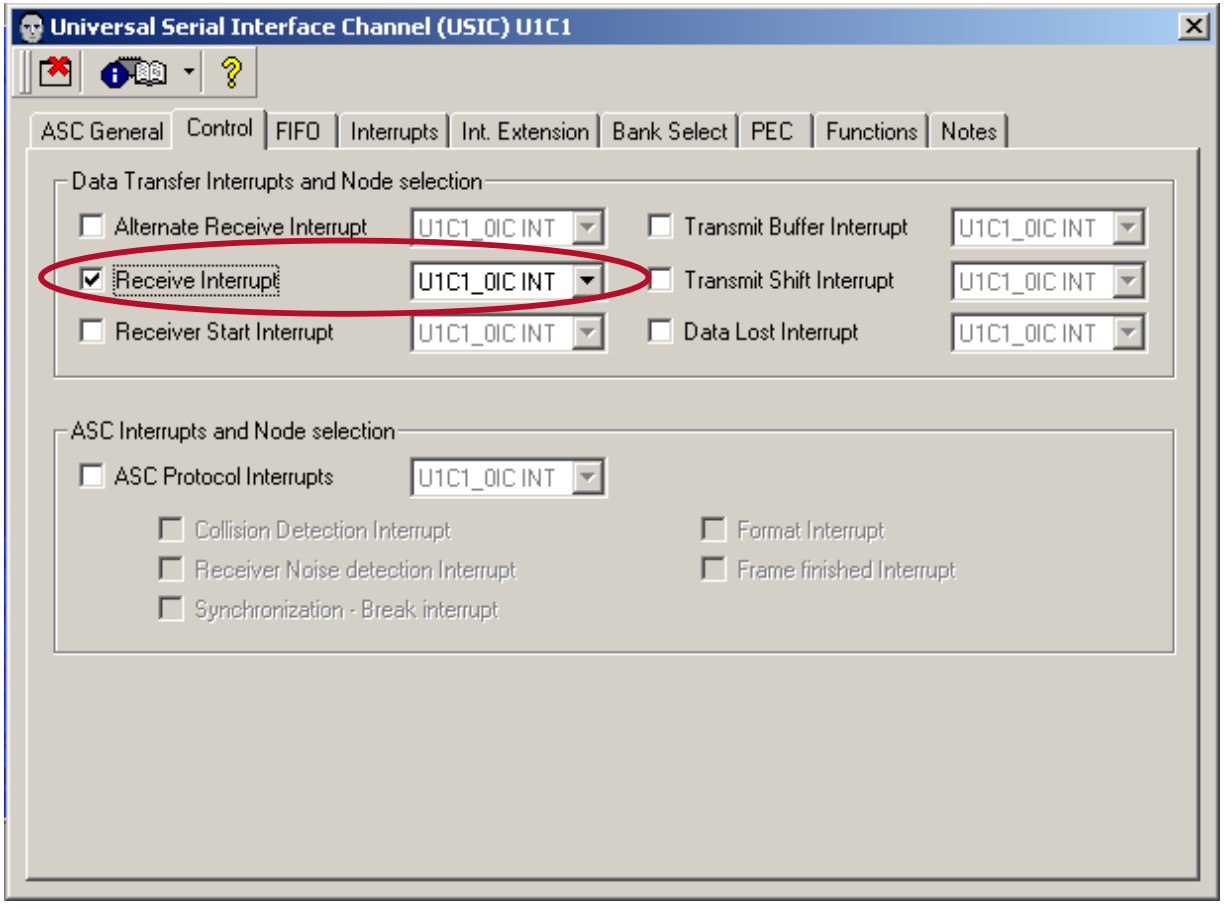
- Click on CH 1 of the USIC 1
- Enable the module
- Select the Pins for TxD and RxD
- Baudrate is 19,2 kBaud
- No parity Bit and 1 Stop Bit is used for the ASC protocol in this example





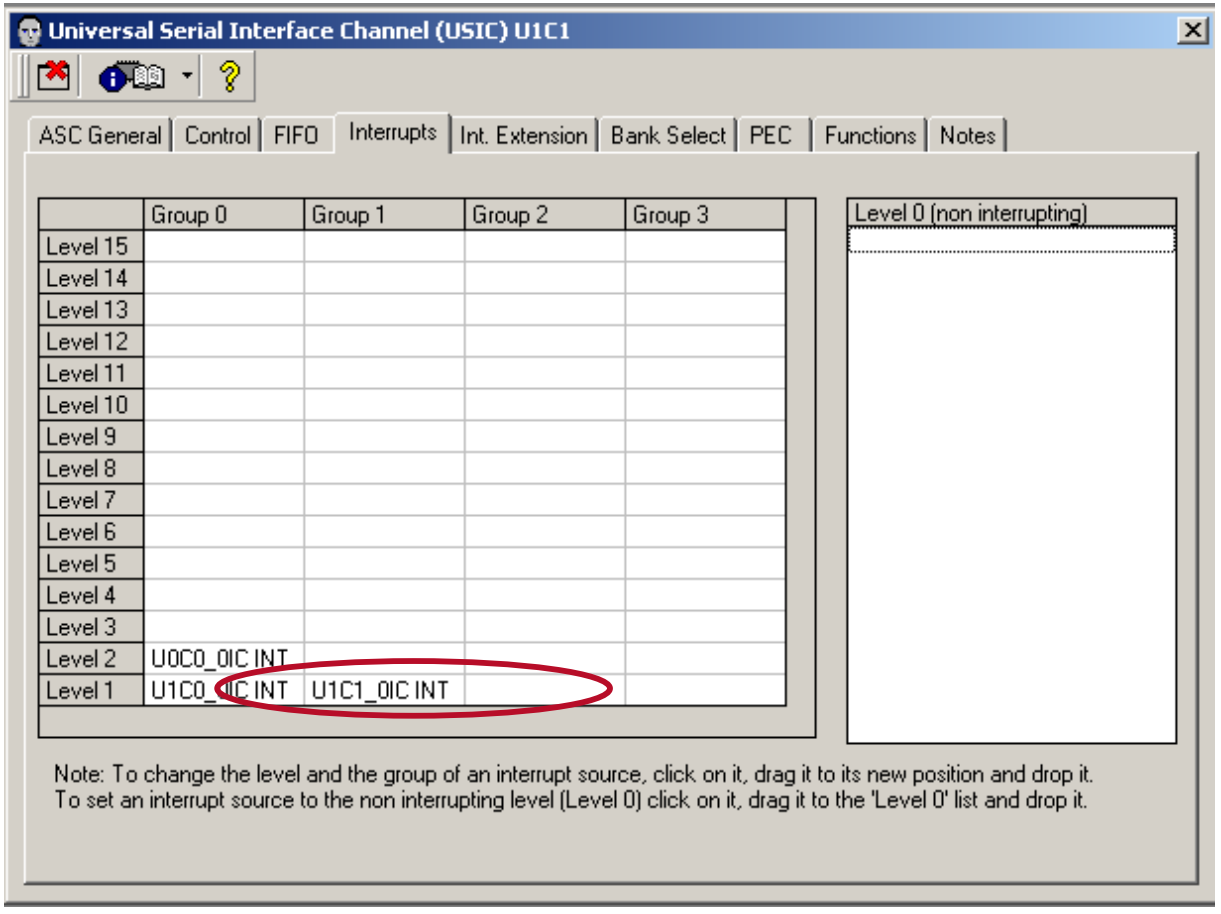
Use DAvE to create your code

- Click on Control of channel 1
- Enable Receive Interrupt



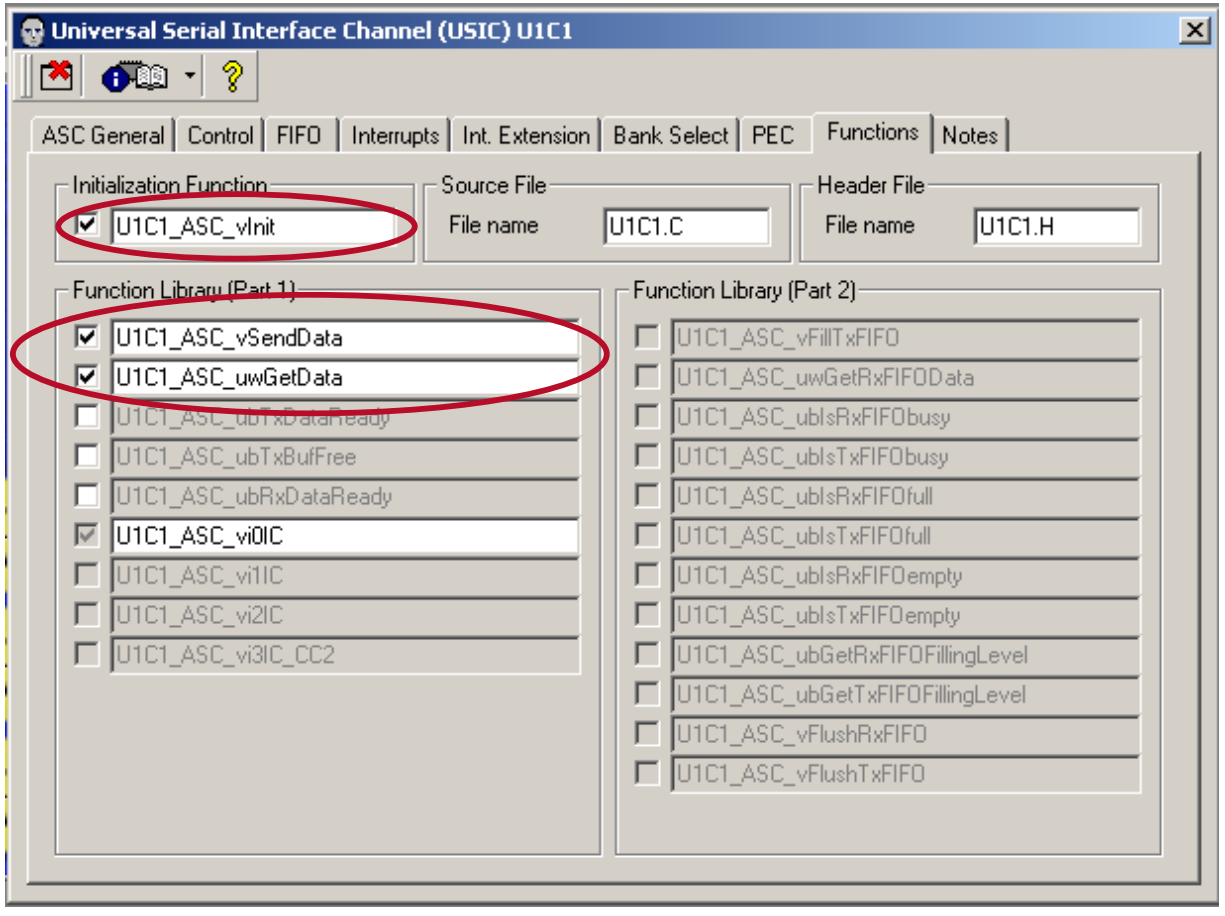
Use DAvE to create your code

- Select the Interrupt Level 1 for the USIC 1 channel 1 interrupt by drag&drop



Use DAvE to create your code

- Enable the USIC 1 Ch 1 ASC init function
- Select additional functions
  - U1C1_ASC_vSendData
  - U1C1_ASC_uwGetData



Use DAvE to create your code

- You can **save** your project as '**HOT_XE164_ASC-Link'** in the **\Source\HOT_XE164_ASC-Link** folder

- **Press** the **Generate** Button: 

DAvE will **generate automatically**
all the necessary **files** for **system**
and **peripheral initialization.**

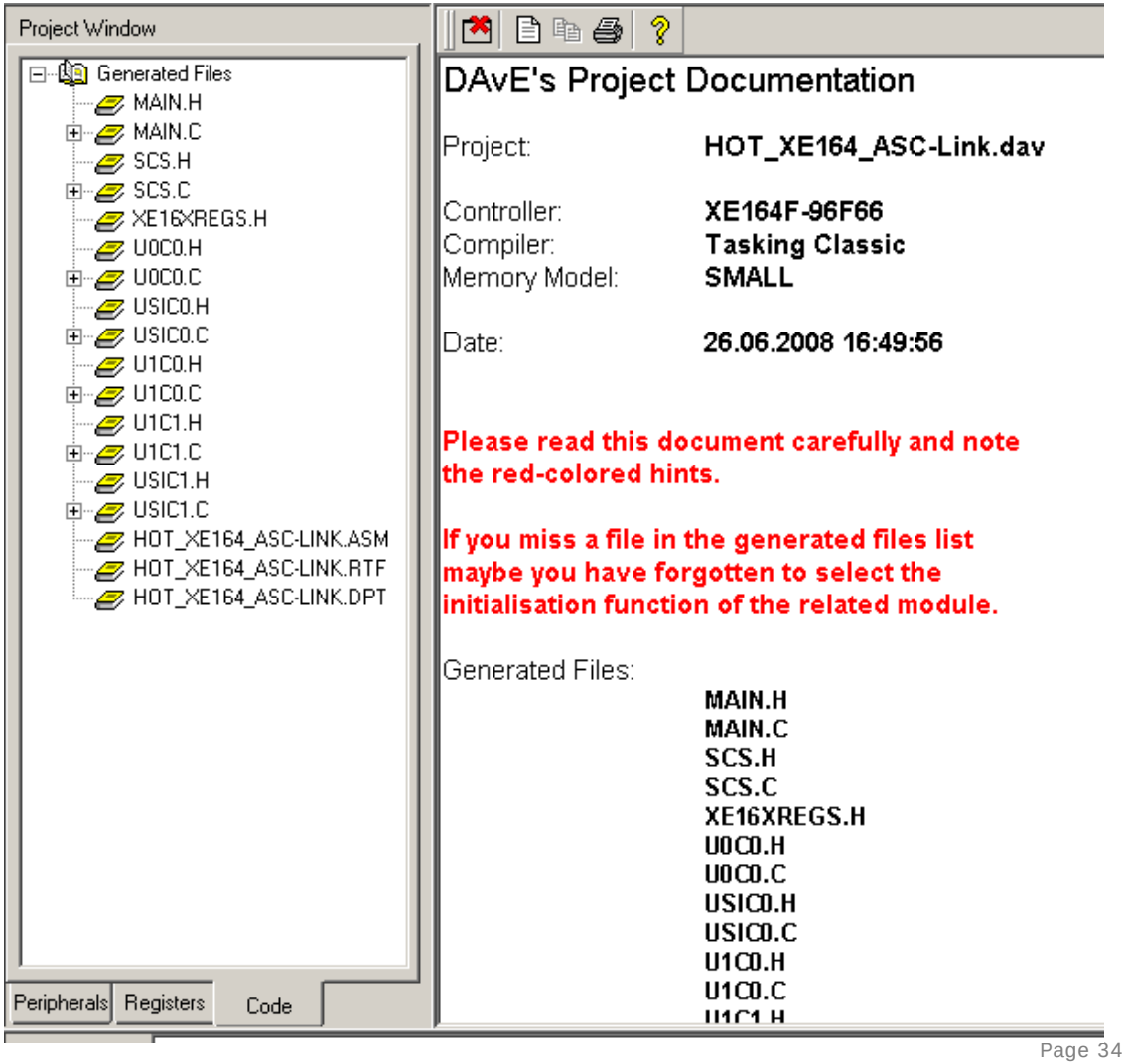


Table of contents

Introduction

Hardware setup

Use DAvE to create HOT_XE164_ASC-Link

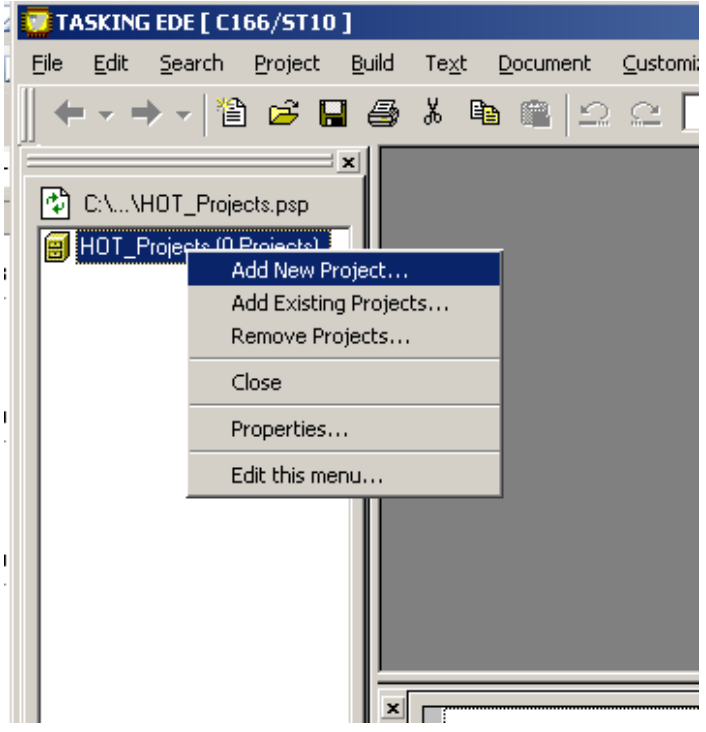
Use Tasking EDE to compile and debug your program

Use Tasking to compile your program

- Launch Tasking

TASKING

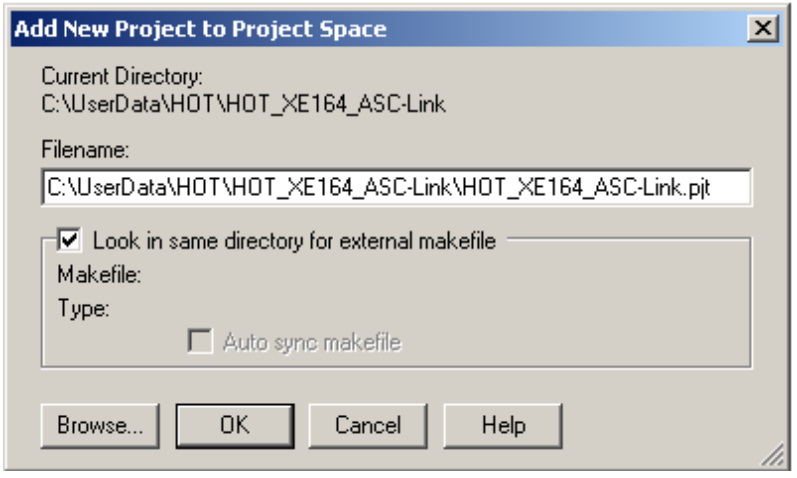
- To integrate all HOT_XE164_ASC-Link files generated by DAvE, first create a new Tasking project
  - Right click on the project space
  - Choose ‘Add New Project’



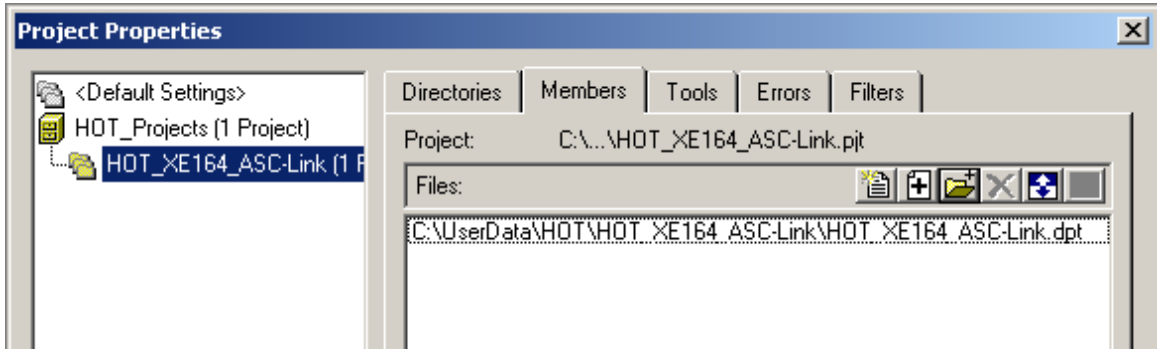
Precondition for this step is a ready-made project space. If no project space is set, check out hint #1.

Use Tasking to compile your program

- Browse for the **\Source\HOT_XE164_ASC-Link** directory and type **HOT_XE164_ASC-Link.pjt**

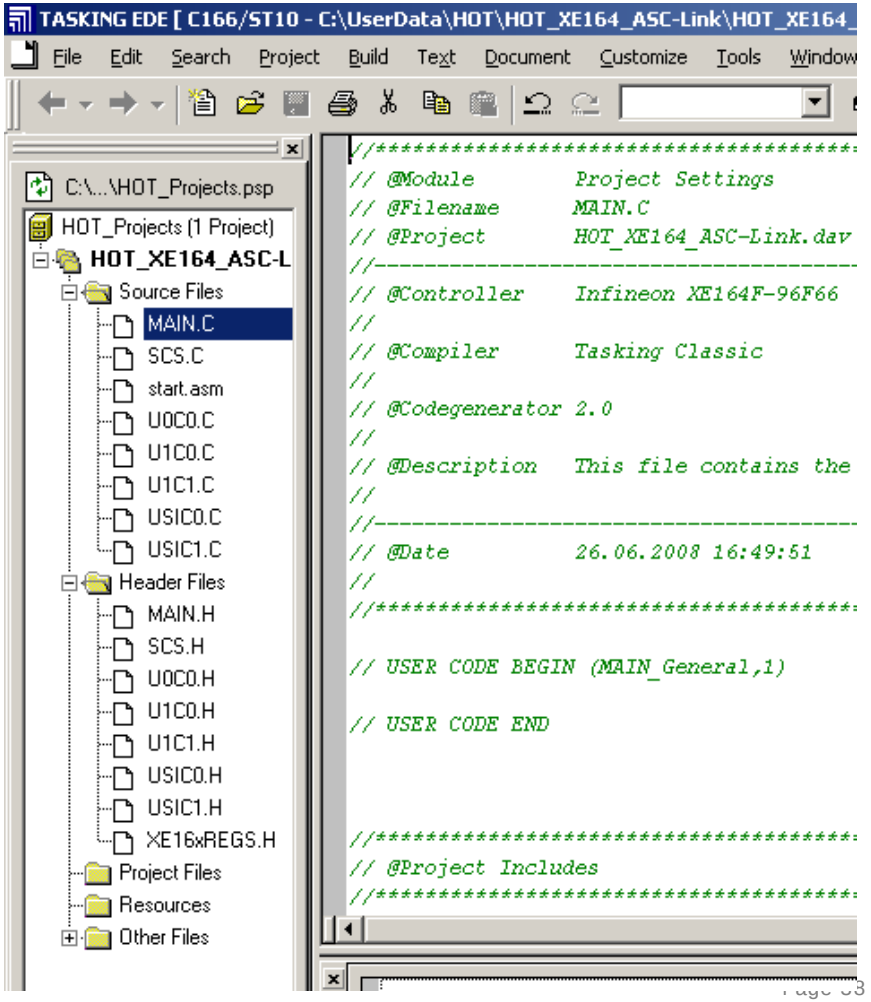


- Add the DAvE generated file **HOT_XE164_ASC-Link.dpt** by clicking on 



Use Tasking to compile your program

- Set HOT_XE164_ASC-Link as the current project by right-click on the project and choose ‘Set as Current Project’
- Click on the ‘Refresh DAvE imported project button’ 
  - All files will be visible



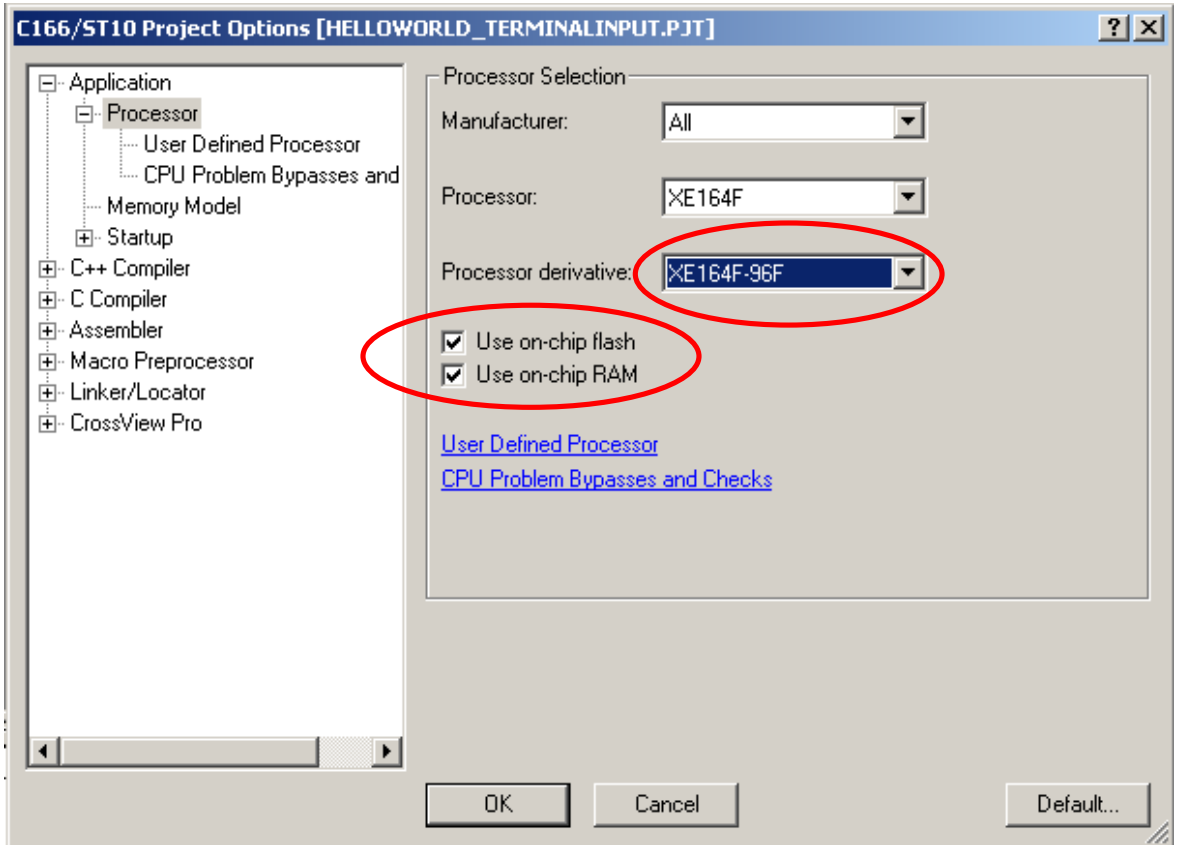
A problem with this step?
Check out hint #2 at the end of this document

Use Tasking to compile your program

- **Complete** the **project options by click on:**

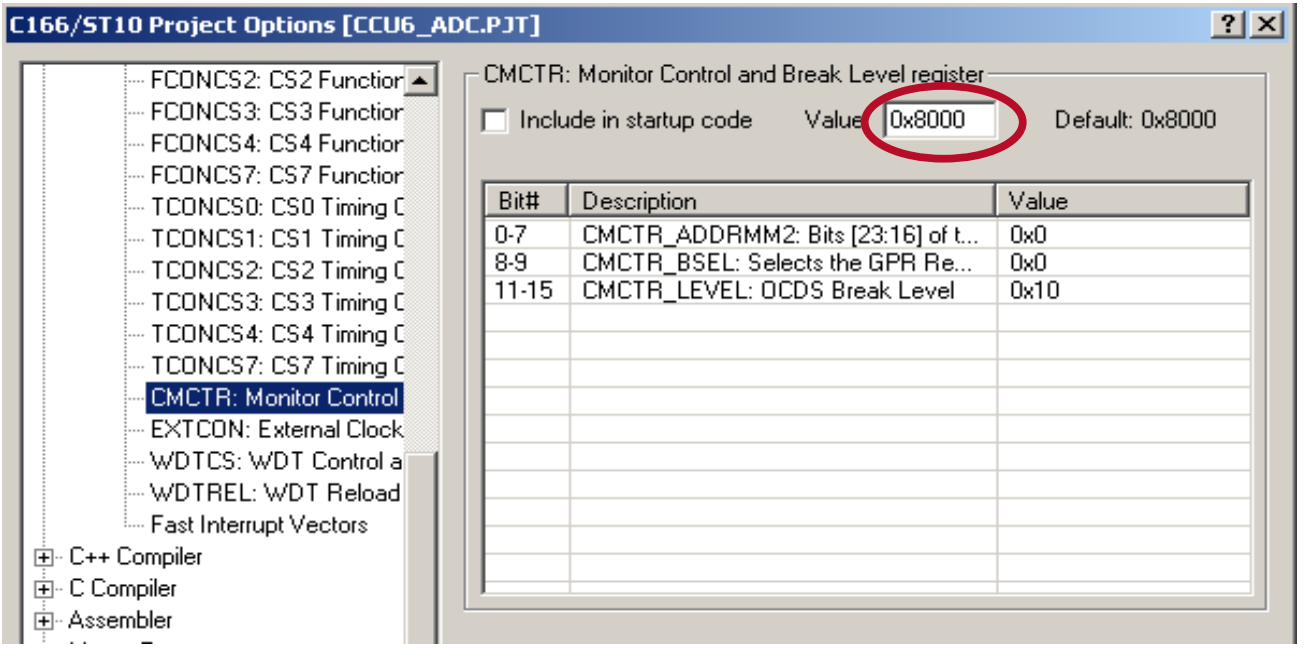


  - **Select the Processor derivative → XE164F-96F**
  - **Enable the on-chip flash and RAM**



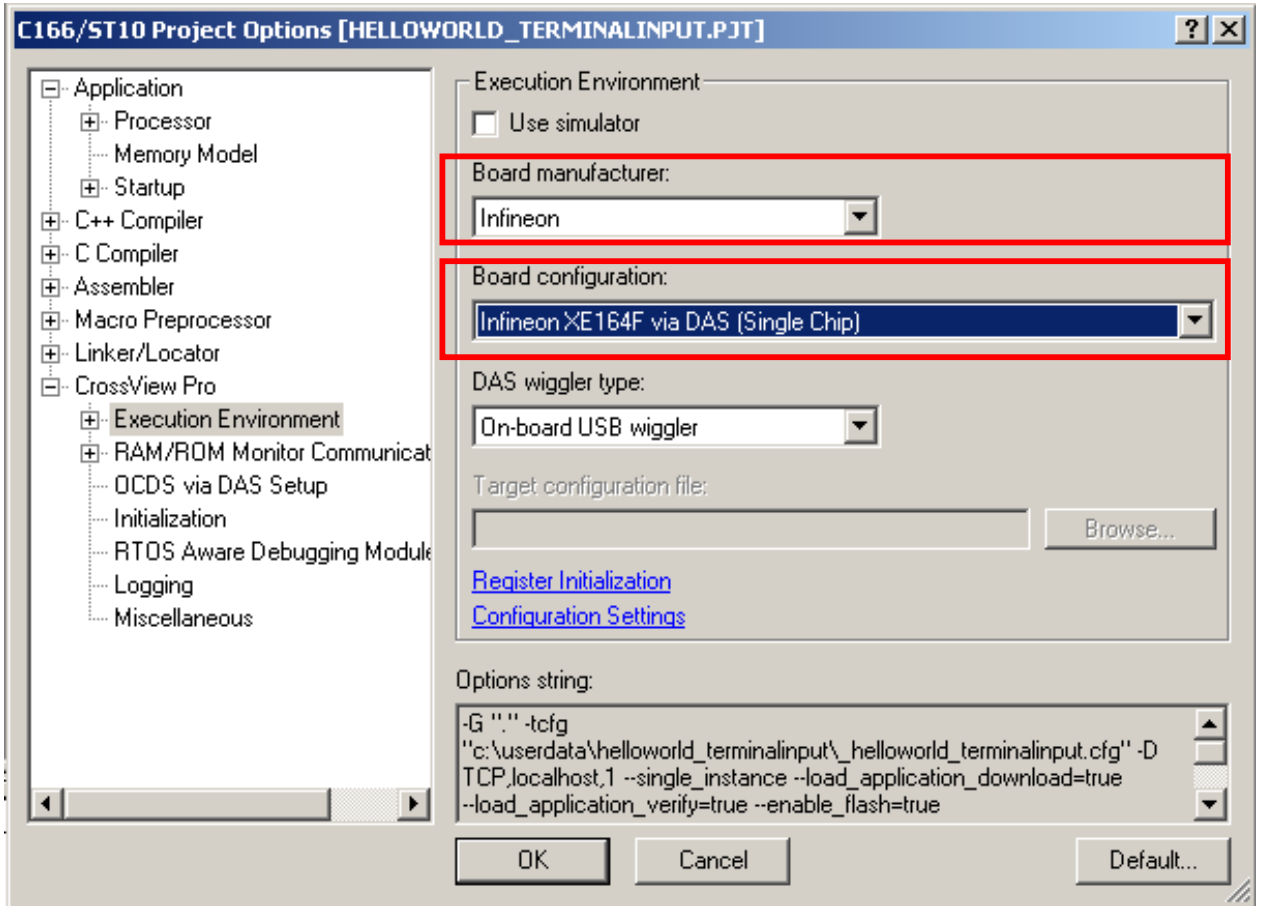
Use Tasking to debug your program

- Check under Application → startup make sure the register CMCTR is set to 0x8000 (after DAvE import the CMCTR register could be set to 0x0000)



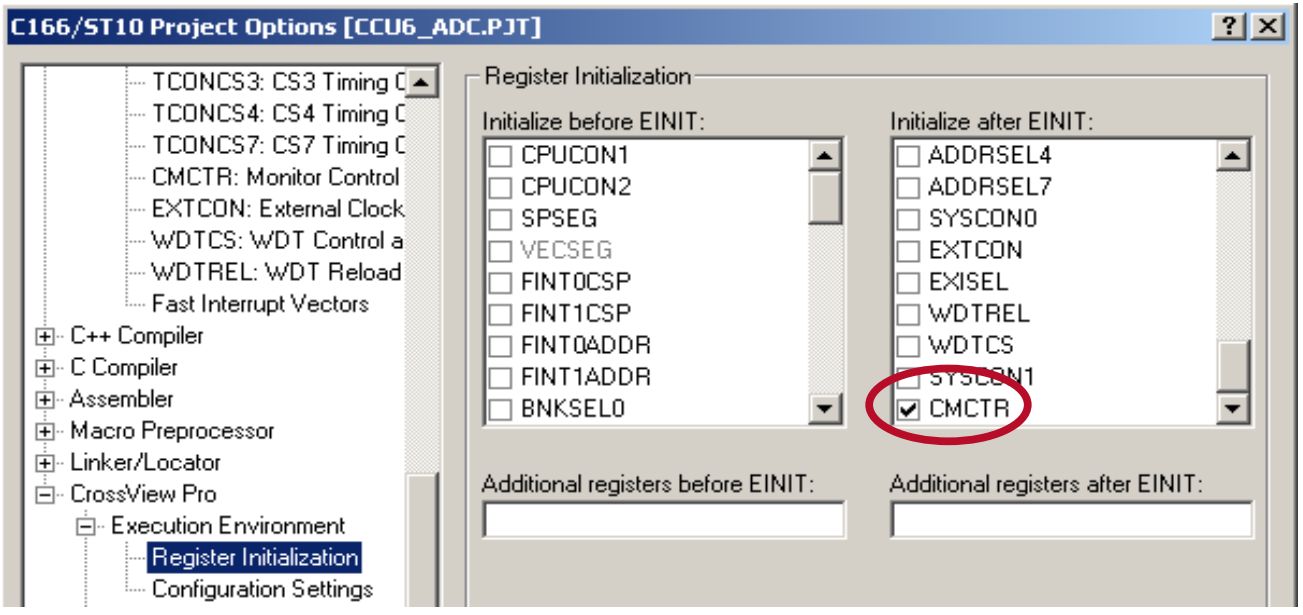
Use Tasking to debug your program

- To Setup the CrossView Debugger Pro following options need to be selected
  - Undo '**Use Simulator**'
  - Choose Infineon as the board manufacturer and select the appropriate board configuration



Use Tasking to debug your program

- Check under **CrossView Pro** register initialization that the register CMCTR is set



Use Tasking to complete your program

- Open the **main** function, **located** in the '**main.c**' file
- Take the DAvE generated function “U0C0_ASC_vSendData” to send a character to the terminal program
- Insert these lines of code previous to the infinite loop in the main function

```
void main(void)
{
  …
 MAIN_vInit();

 // USER CODE BEGIN (Main,3)
 U0C0_ASC_vSendData('\n');
 U0C0_ASC_vSendData('\r');
 U0C0_ASC_vSendData('H');
 U0C0_ASC_vSendData('O');
 U0C0_ASC_vSendData('T');
 U0C0_ASC_vSendData('_');
 U0C0_ASC_vSendData('X');
 U0C0_ASC_vSendData('E');
 U0C0_ASC_vSendData('1');
 U0C0_ASC_vSendData('6');
 U0C0_ASC_vSendData('4');
 U0C0_ASC_vSendData('_');
 U0C0_ASC_vSendData('A');
 U0C0_ASC_vSendData('S');
 U0C0_ASC_vSendData('C');
 U0C0_ASC_vSendData('-');
 U0C0_ASC_vSendData('L');
 U0C0_ASC_vSendData('i');
 U0C0_ASC_vSendData('n');
 U0C0_ASC_vSendData('k');
 U0C0_ASC_vSendData('\n');
 U0C0_ASC_vSendData('\r');
 // USER CODE END

 while(1)
```

Use Tasking to complete your program

- Take the DAvE generated function “U0C0_ASC_uwGetData” to read a character received from the terminal program
- Use the function “U1C0_ASC_vSendData” to transmit the character (received from the terminal program) from the ASC interface realized by USIC1 CH0 to the ASC interface realized by USIC1 CH1
- Insert this line of code in the USIC0 channel 0 interrupt service routine located in the file **U0C0.C**

```
_interrupt(U0C0_0INT)  void U0C0_ASC_vi0IC(void)
{

  // USER CODE BEGIN (ASC0IC,2)

  // USER CODE END

  if (U0C0_PSR & 0x4000)        // Receive interrupt flag
  {

  // USER CODE BEGIN (ASC0IC,4)
  U1C0_ASC_vSendData(U0C0_ASC_uwGetData());
  // USER CODE END

  U0C0_PSCR   |= 0x4000;        // clear PSR_RIF
  }

  // USER CODE BEGIN (ASC0IC,15)
```

Use Tasking to complete your program

- Go to the receive interrupt service routine of USIC1 CH1 located in the '**U1C1.C**' file
- Use the function "U1C1_ASC_uwGetData" to read the character received from the ASC interface realized by USIC1 CH0
- This character first should be sent to the terminal program and second back to the ASC interface realized by USIC1 CH0
- To ease-up data handling define the variable U1C1_rxCharacter

```
// USER CODE BEGIN (ASC0IC,1)
uword U1C1_rxCharacter;
// USER CODE END

_interrupt(U1C1_0INT)  void U1C1_ASC_vi0IC(void)
{

  // USER CODE BEGIN (ASC0IC,2)

  // USER CODE END

  if (U1C1_PSR & 0x4000)        // Receive interrupt flag
  {

  // USER CODE BEGIN (ASC0IC,4)
  U1C1_rxCharacter = U1C1_ASC_uwGetData();
  U0C0_ASC_vSendData(U1C1_rxCharacter);
  U1C1_ASC_vSendData(U1C1_rxCharacter);
  // USER CODE END

  U1C1_PSCR   |= 0x4000;        // clear PSR_RIF
  }

  // USER CODE BEGIN (ASC0IC,15)
```

Use Tasking to compile your program

- Read the character received from the ASC interface realized by USIC1 CH1 with the function “U1C0_ASC_uwGetData” and transmit the character to the terminal program
- Insert this line of code in the receive interrupt service routine of USIC1 CH0 located in **U1C0.C**
- Finally compile the program by pushing the button 

```
_interrupt(U1C0_0INT)  void U1C0_ASC_vi0IC(void)
{

  // USER CODE BEGIN (ASC0IC,2)

  // USER CODE END

  if (U1C0_PSR & 0x4000)        // Receive interrupt flag
  {

  // USER CODE BEGIN (ASC0IC,4)
  U0C0_ASC_vSendData(U1C0_ASC_uwGetData());
  // USER CODE END

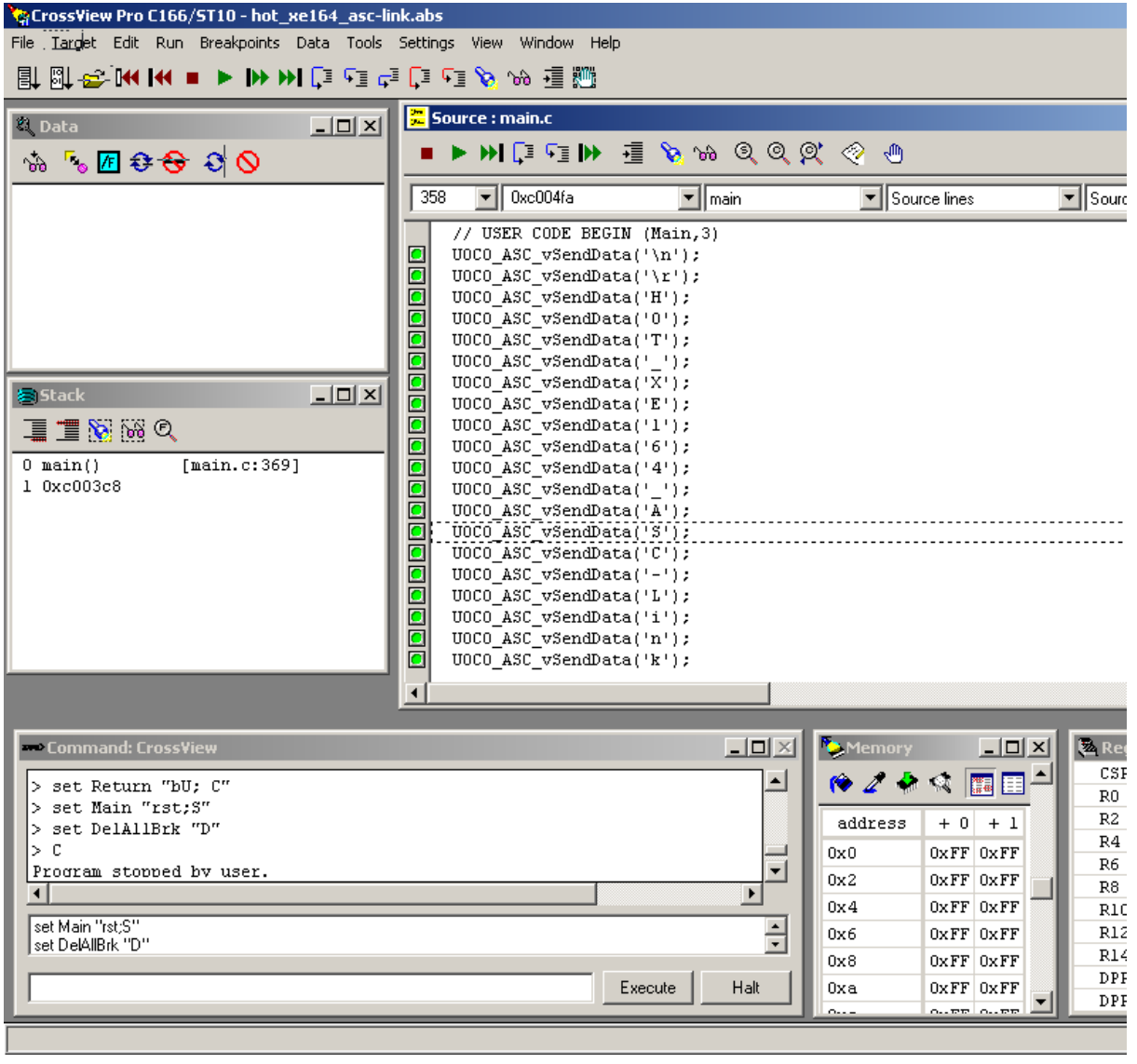
  U1C0_PSCR   |= 0x4000;        // clear PSR_RIF
  }

  // USER CODE BEGIN (ASC0IC,15)
```

A problem with this step? Check out hint #3 at the end of this document

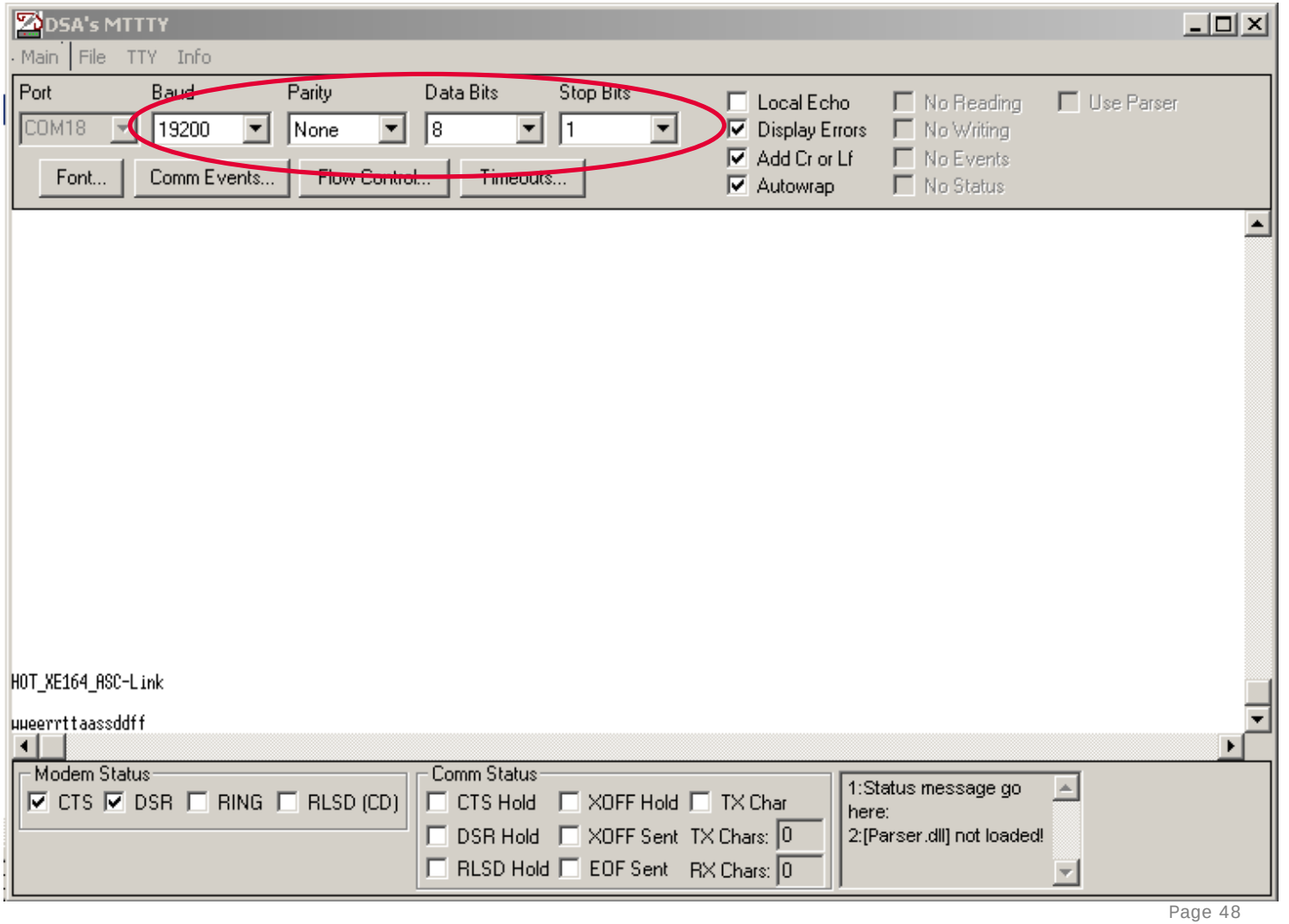
Use Tasking to debug your program

- **To start the USB Debugger** press the button 
- **To start the program** press the button 



Use a Terminal Program to see your application running

- Open a Terminal Program and use the same ASC settings you made in DAvE
- Type in some characters and watch your keybord input on the Terminal Program





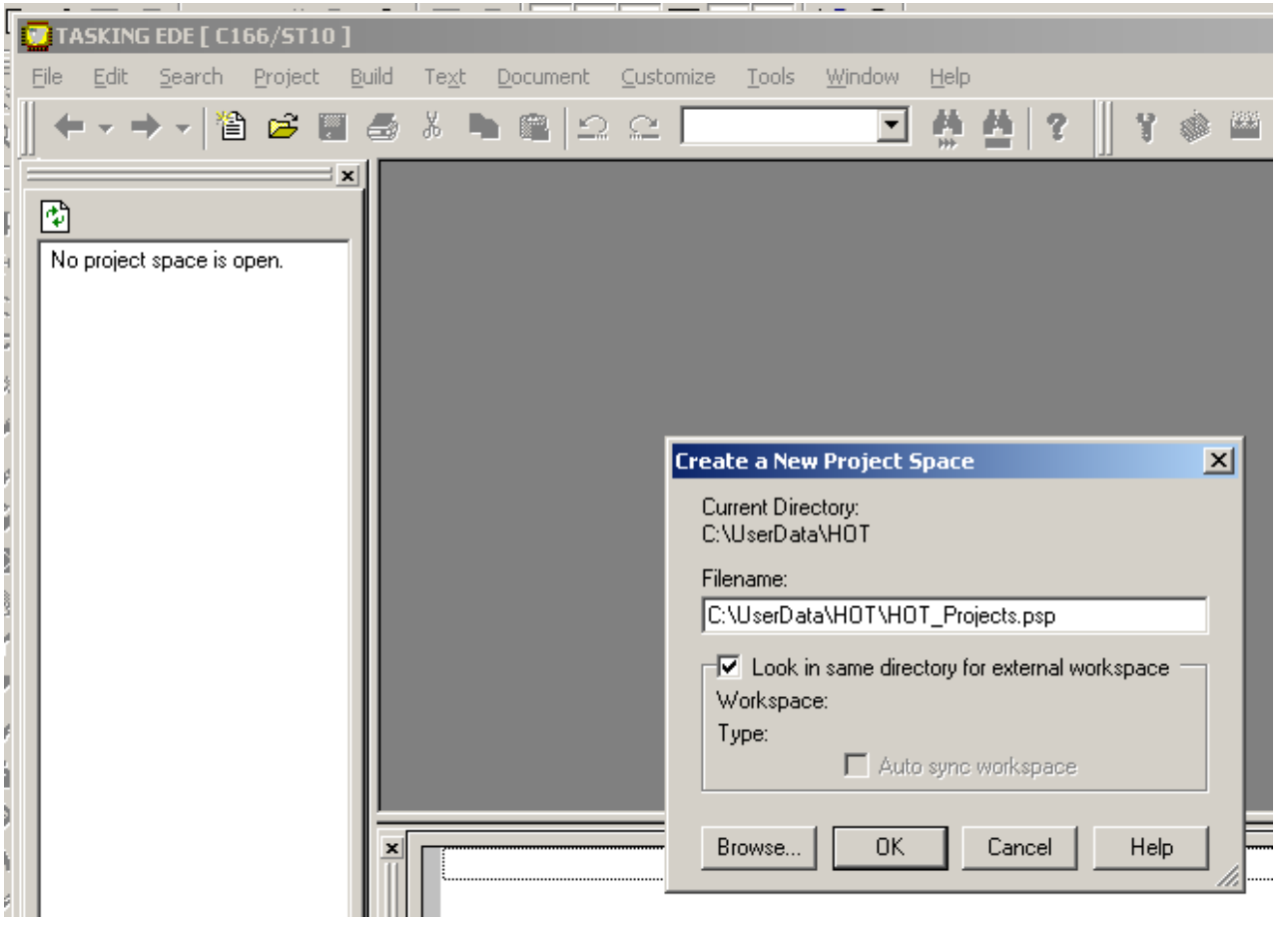
Never stop thinking

Hint #1

- Launch Tasking

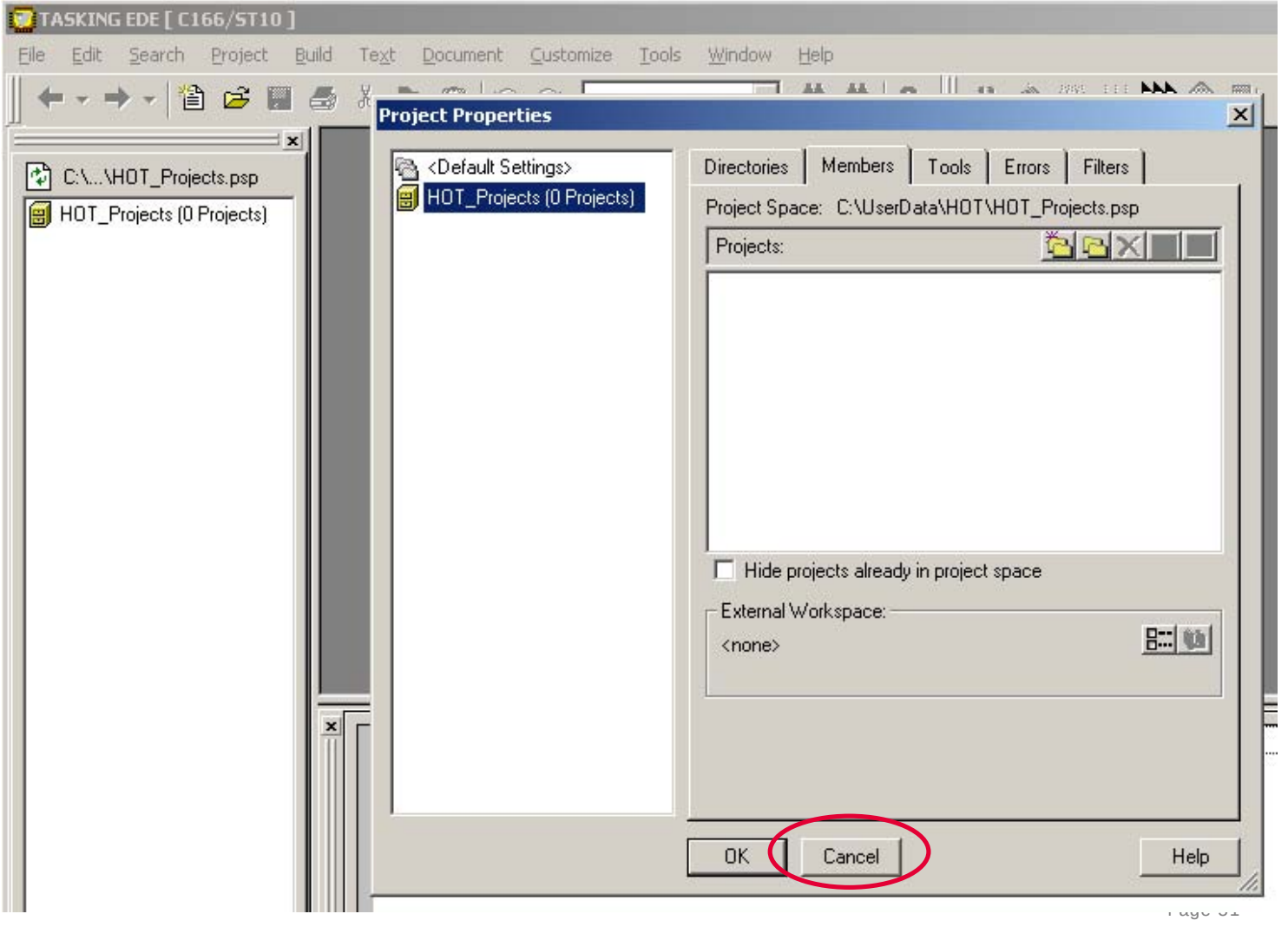
TASKING

- Goto **File** -> **New Project Space**...
- Browse for your target directory and choose a name, e.g. '**HOT_Projects.psp**'



Hint #1

- Integration of projects can be done after AutoCode generation with DAvE,
  so choose **cancel**



Hint #2

- In Tasking, when you click on the **Refresh DAvE imported project** button**,** an error message might appear :





- That means that you do not have the **version 2.0** of the **DAvE XE16xx Series Dip**. To install it, please follow the steps in the **DAvE_XE16xx_Series_Dip_Installation_Procedure_v20.pdf**

Hint #3

- This is a known issue of Tasking Classic v8.7r1 FREE VERSION and DAvE XE164 Dip file version 1.0
- Use Tasking Classic v8.7r2 and/or DAvE XE164 Dip file version 2.0 and higher or follow the workaround below
  - Define the _inline functions used in the System Configuration file (SCS.C)

```
//**************************************************************************
// @Project Includes
//**************************************************************************

#include "MAIN.H"

// USER CODE BEGIN (SCS_General,2)
#ifndef _inline
#define _inline
#endif
// USER CODE END
```