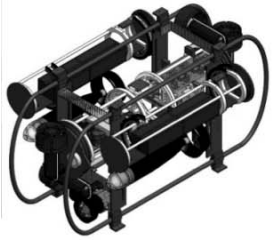
IMDL Software Series

Atmel/AVR Studio 7
Atmel AVR Xmega Code
Getting Started with ASF,
I/O Ports & USART with the
OOB DAQ Board
&
Arduino MEGA2560 Serial I/O

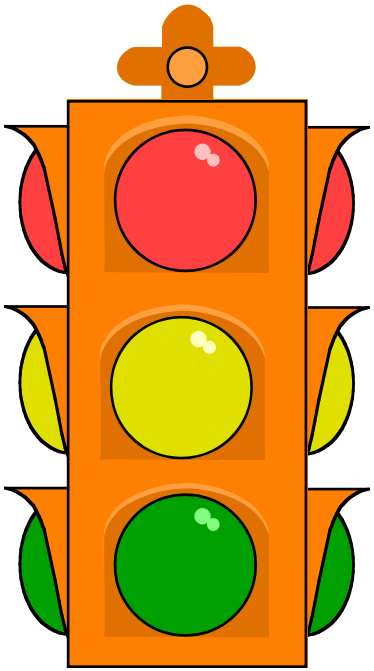
A. A. Arroyo

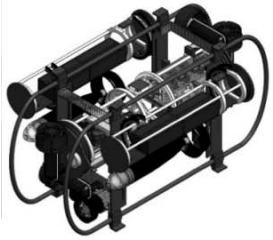


IMDL Software Series

Today's Menu

- Getting Started with Atmel ASF in AVR Studio 7 for the DAQ Board
- General Purpose I/O Pins
- Serial I/O



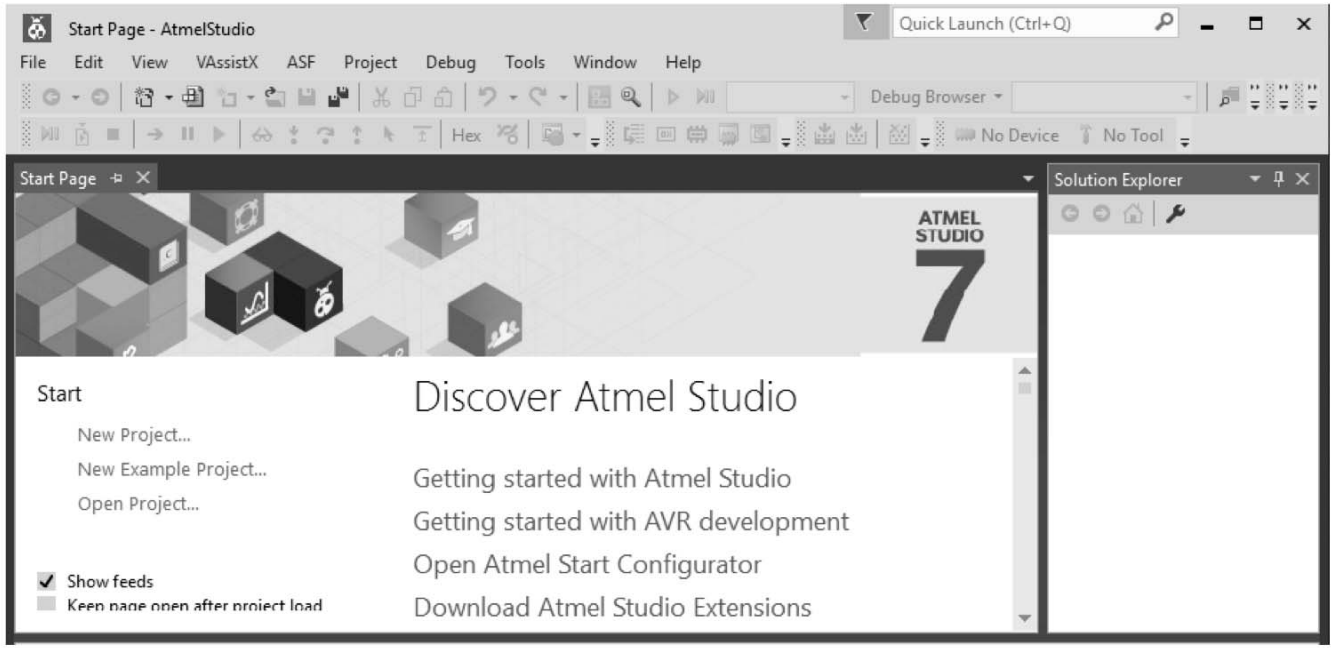


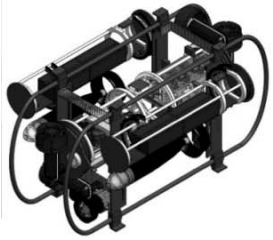
IMDL Software Series

AVR Studio 7 & Atmel ASF

Creating new ASF Projects using AVR Studio 7 for the DAQ Board(s)

Step 1: Select Open New Project on the AVR Studio 7 Opening Screen or alternatively select New>Project from the File Menu (ctrl+shift+n)

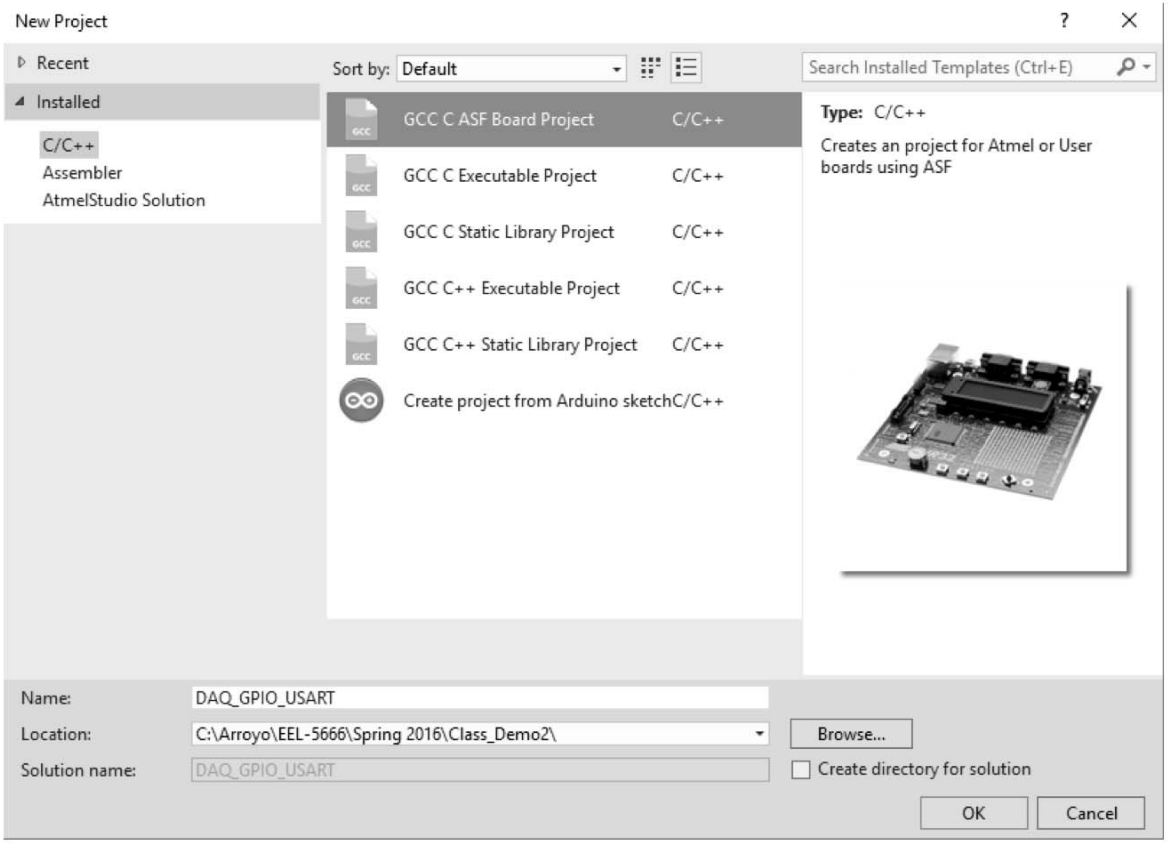


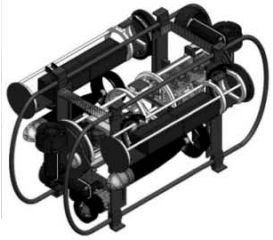


IMDL Software Series

AVR Studio 7 & Atmel ASF

Step 2: Select C/C++, enter a name for your project and select a directory for your project in the New Project Window. Uncheck the *Create Directory for Solution* check box. Hit OK when ready.

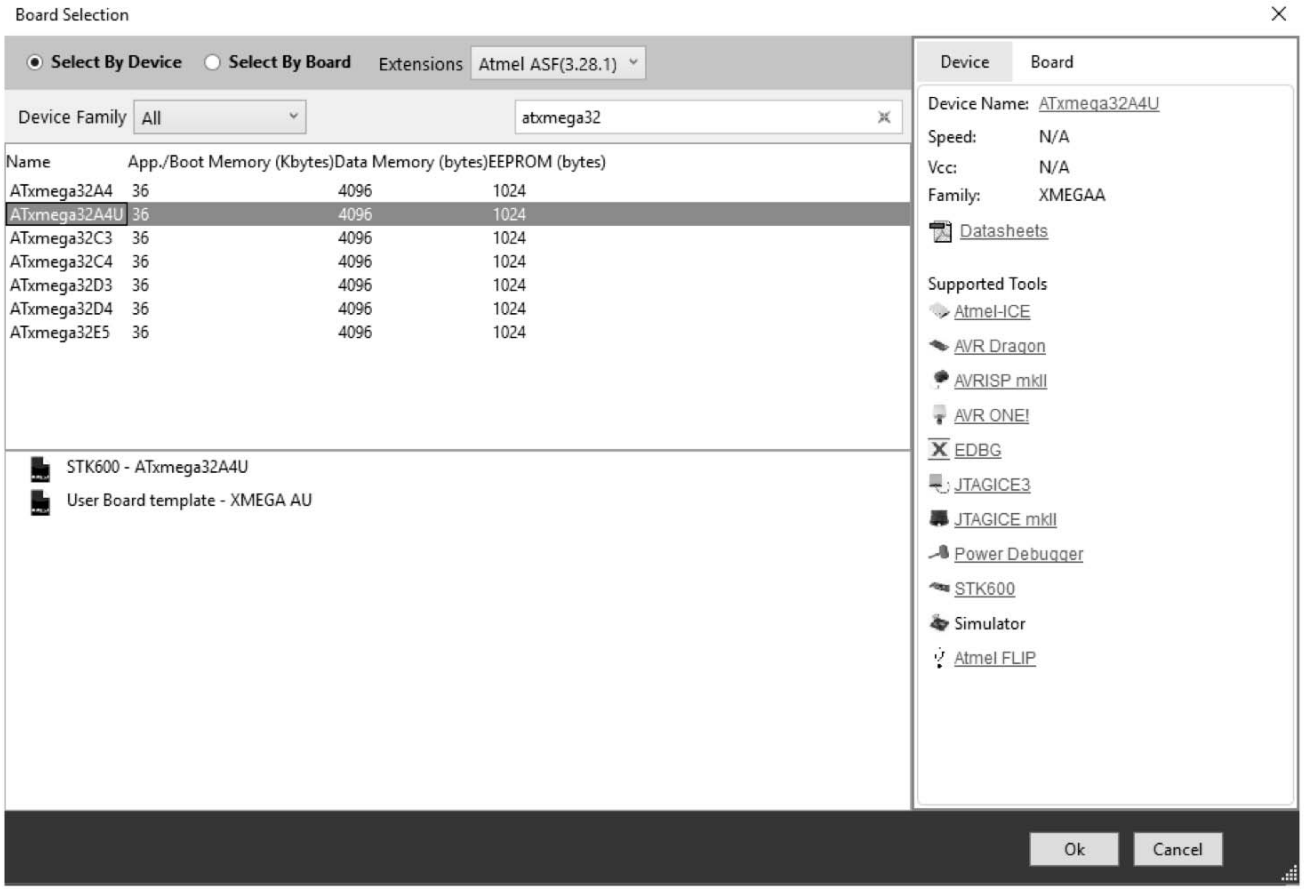


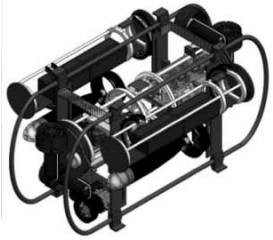


IMDL Software Series

AVR Studio 7 & Atmel ASF

Step 3: Select ATxmega32A4U (used by DAQ) in the device selection window and User Board Template XMEGA AU. Hit OK.

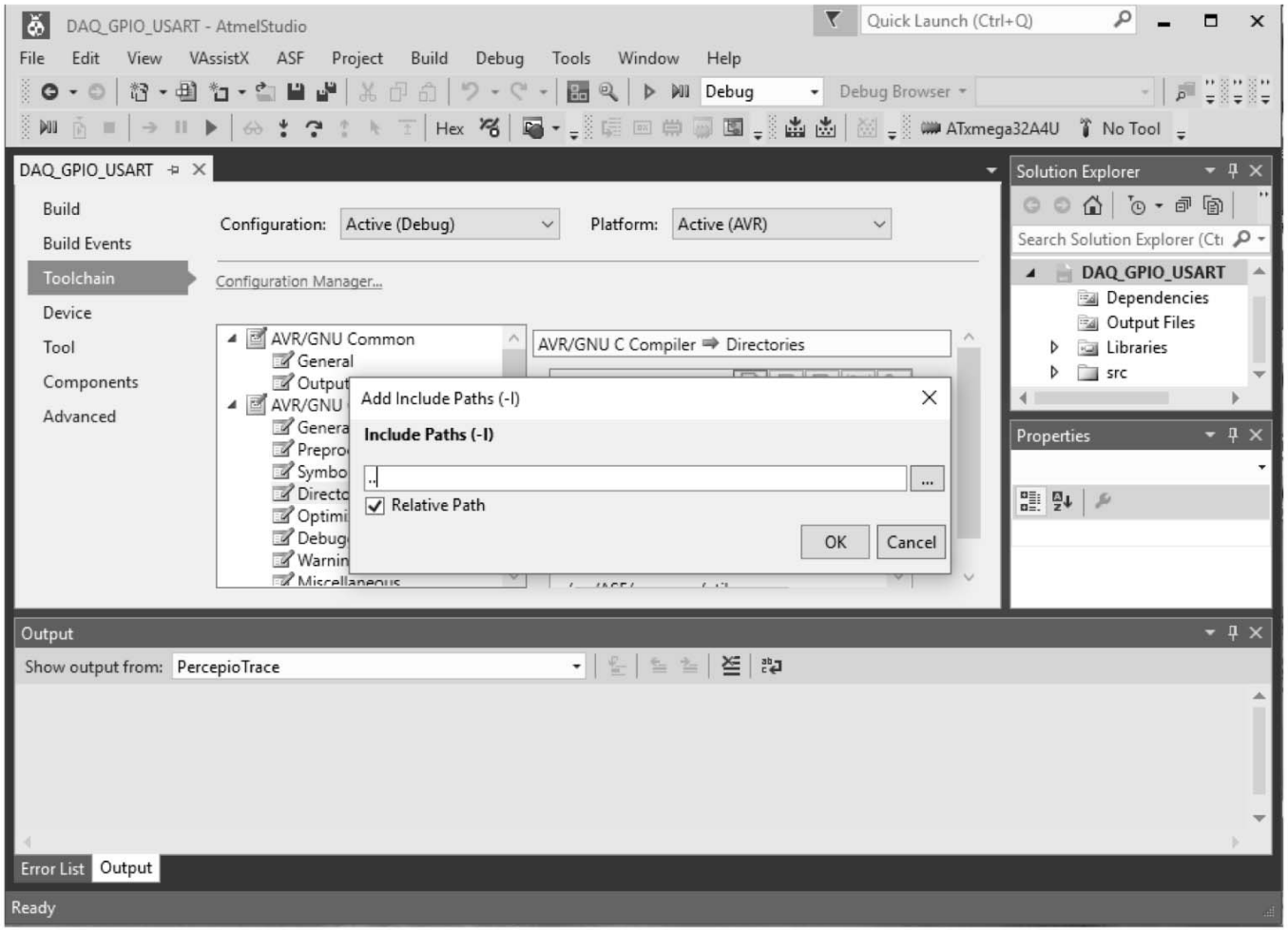


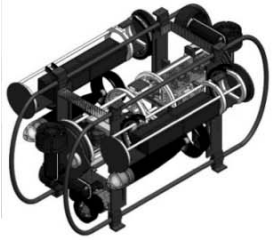


IMDL Software Series

AVR Studio 7 & Atmel ASF

Step 4: Add path ".." Toolchain Directories section. Hit OK.

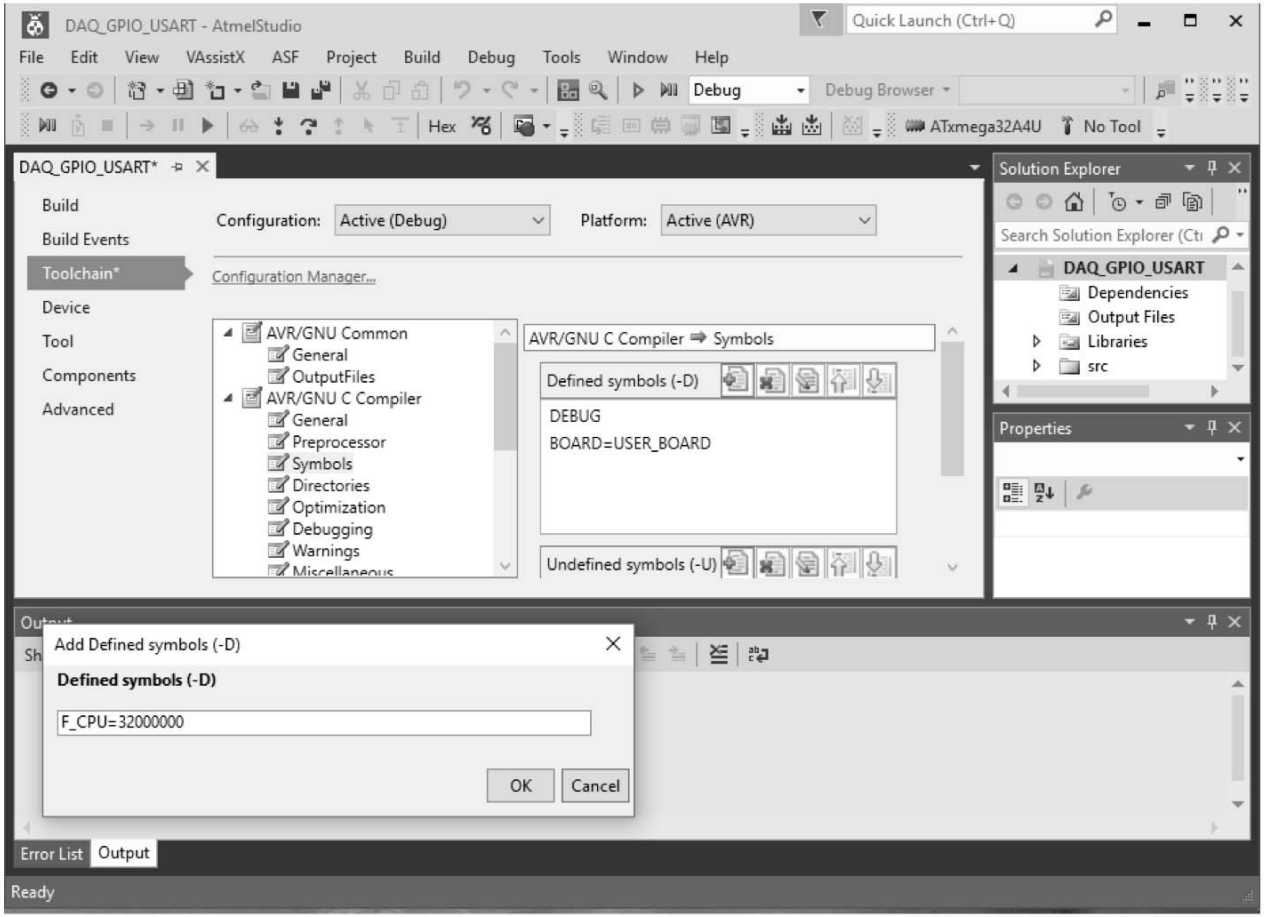


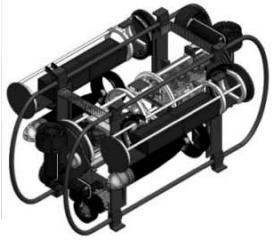


IMDL Software Series

AVR Studio 7 & Atmel ASF

Step 5: Add symbol "F_CPU=32000000" to the Toolchain Symbols section. Hit OK

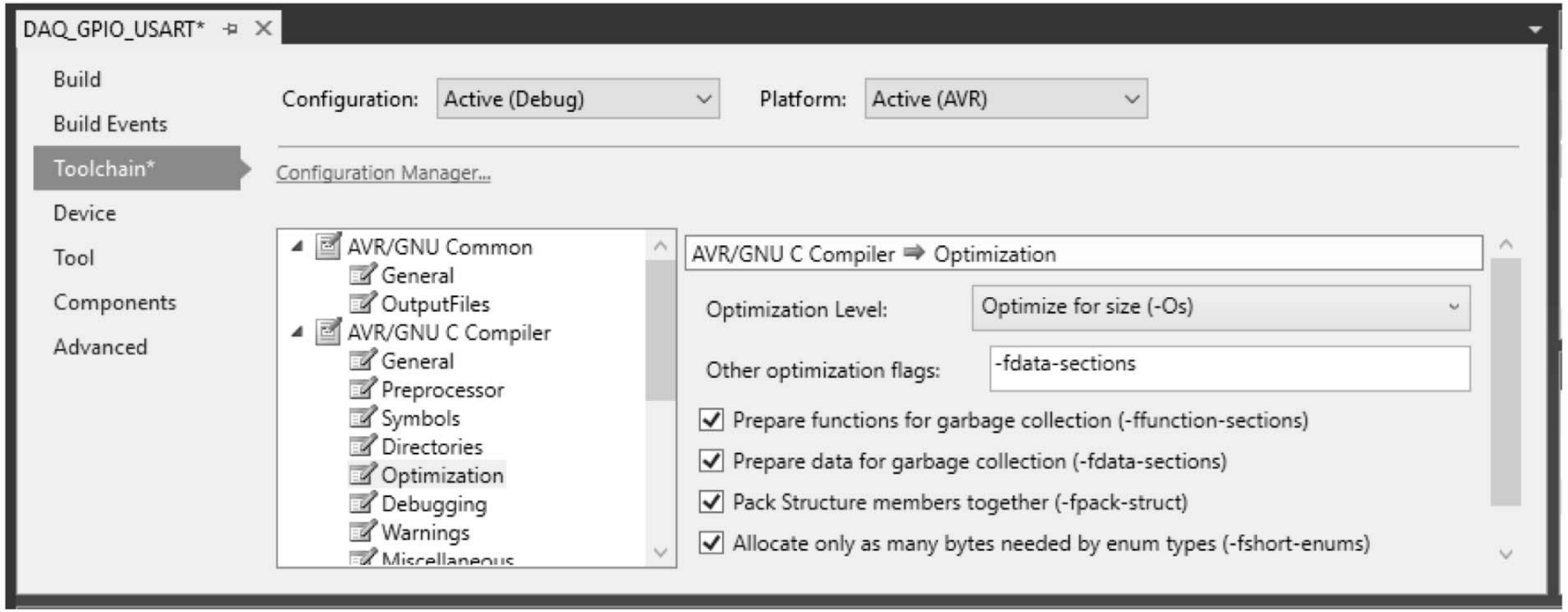


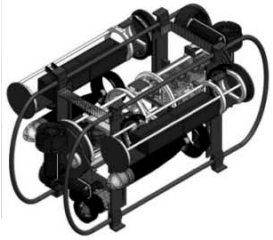


IMDL Software Series

AVR Studio 7 & Atmel ASF

Step 6: Add Compiler Optimization to Os as Shown. Hit OK

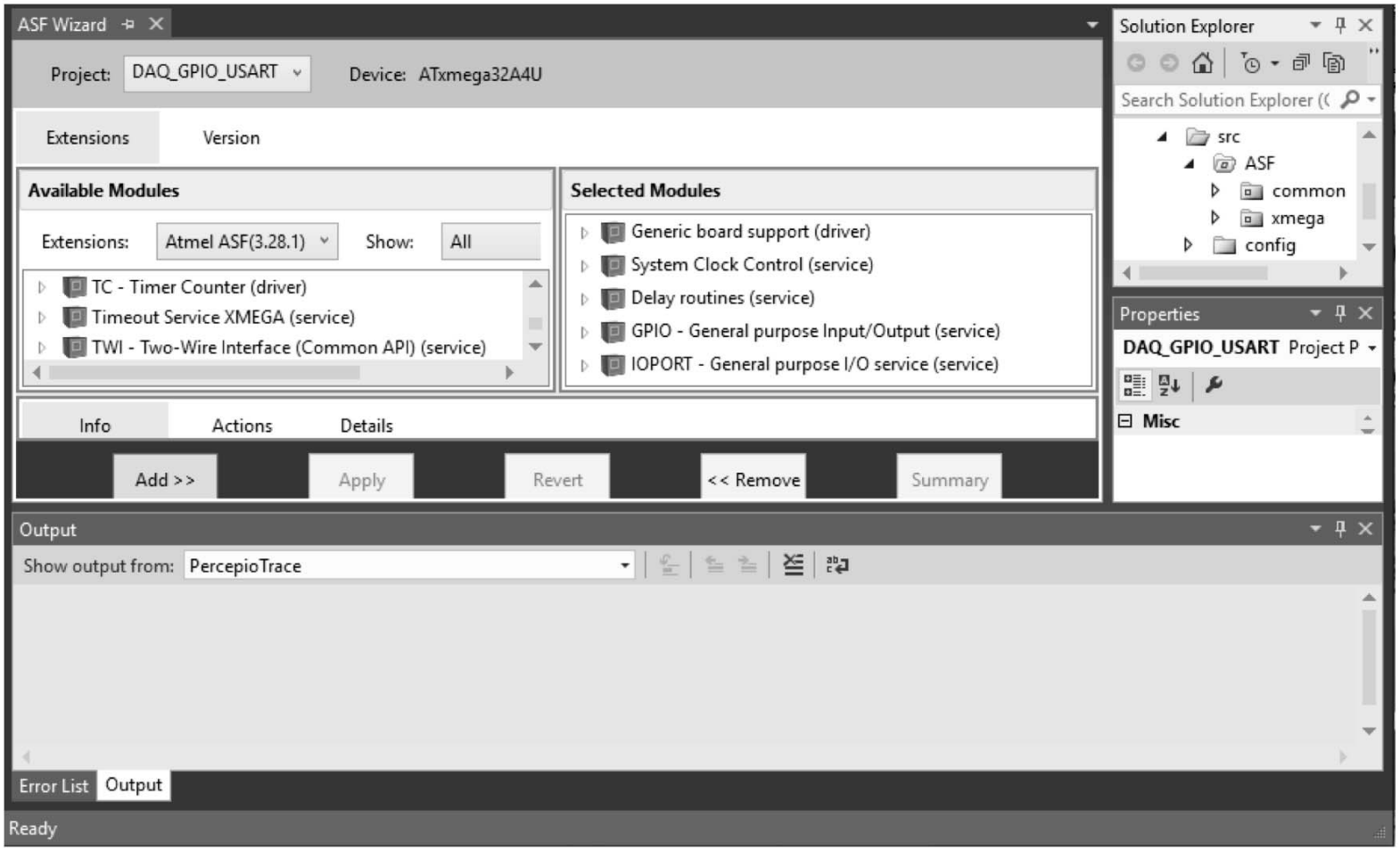


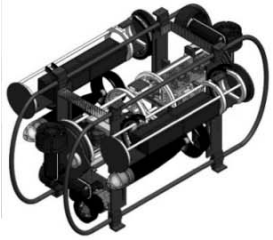


IMDL Software Series

AVR Studio 7 & Atmel ASF

Step 7: Using ASF Wizard Select Desired ASF Modules & Hit Apply

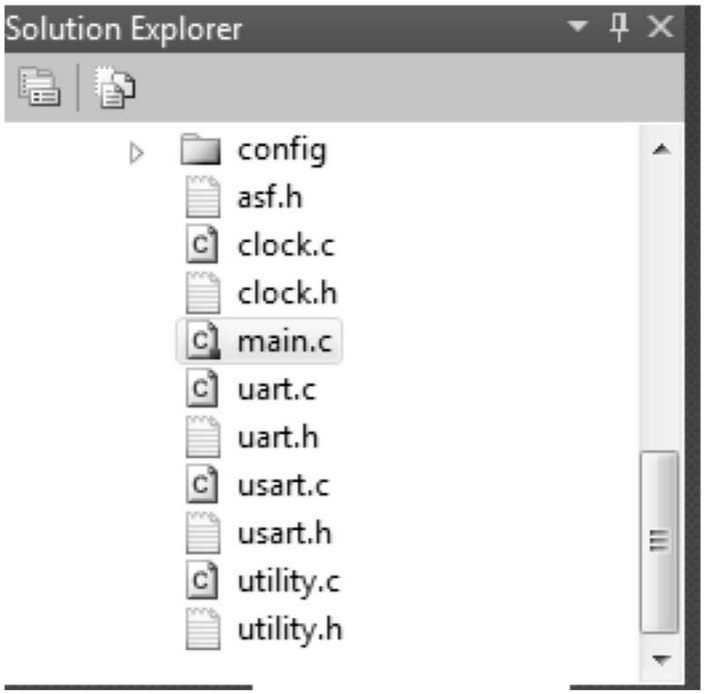


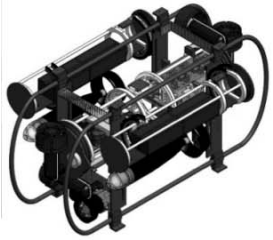


IMDL Software Series

AVR Studio 7 & Atmel ASF

Step 8: Import clock.c, clock.h, utility.c, utility.h which are used by init.c from Tim's / Lab DAQ Project.



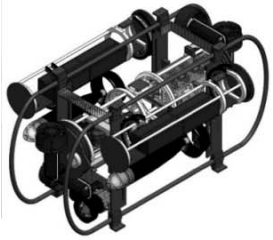


AVR Studio 7 & Atmel ASF

Step 9: Start writing your code by modifying main.c and creating any ".c" and ".h" files as appropriate.

```
#include <asf.h>
#include <avr/io.h>
#include <stdio.h>

int main(void) {
  board_init();      // initialize the DAQ Board
  // TODO place your initialization code here
  while(1) {                    // Main Loop
  //TODO write your main loop code here
  }
}
```



AVR Studio 7 & Atmel ASF

- The first example will use PortD pins to toggle external LEDs connected to pins PD0 & PD1
- Import System Clock, Delays, GPIO and IOPORT services from the ASF Library
- ~~In directory src/ASF/common/services/ioport/xmega in file ioport.h, after the macro in line 46:~~

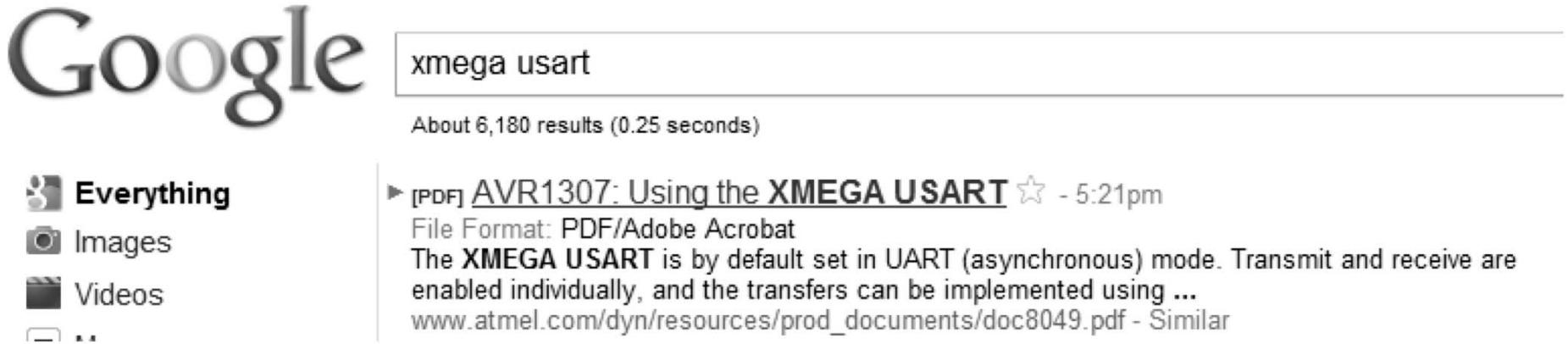
~~#define IOPORT_CREATE_PIN(port, pin) ((IOPORT_## port) * 8 + (pin))~~

~~add the following macro (for convenience):~~

~~#define IOPORT_CREATE_ID(port) ((IOPORT_## port) + 0)~~

IMDL Software Series
Remember to Find Examples!

- Type "xmega usart" into Google



- The first result is a nice 7-page document explaining EXACTLY how to set up the serial USART on the Xmega. The source code for this example is easily found on the internet.

IMDL Software Series
USART

21. USART

21.1 Features

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- Enhanced Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun and Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode
- Master SPI mode, Three-wire Synchronous Data Transfer
  - Supports all four SPI Modes of Operation (Mode 0, 1, 2, and 3)
  - LSB First or MSB First Data Transfer (Configurable Data Order)
  - Queued Operation (Double Buffered)
  - High Speed Operation ($f_{XCK,max} = f_{PER}/2$)
- IRCOM Module for IrDA compliant pulse modulation/demodulation

IMDL Software Series
USART

Table 21-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Conditions	Equation for Calculation Baud Rate[1]	Equation for Calculation BSEL Value
Asynchronous Normal Speed mode (CLK2X = 0)	BSCALE ≥ 0 $f_{BAUD} \le \frac{f_{PER}}{16}$	$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE} \cdot 16(BSEL + 1)}$	$BSEL = \frac{f_{PER}}{2^{BSCALE} \cdot 16 f_{BAUD}} - 1$
	BSCALE < 0 $f_{BAUD} \le \frac{f_{PER}}{16}$	$f_{BAUD} = \frac{f_{PER}}{16((2^{BSCALE} \cdot BSEL) + 1)}$	$BSEL = \frac{1}{2^{BSCALE}}\left(\frac{f_{PER}}{16 f_{BAUD}} - 1\right)$
Asynchronous Double Speed mode (CLK2X = 1)	BSCALE ≥ 0 $f_{BAUD} \le \frac{f_{PER}}{8}$	$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE} \cdot 8 \cdot (BSEL + 1)}$	$BSEL = \frac{f_{PER}}{2^{BSCALE} \cdot 8 f_{BAUD}} - 1$
	BSCALE < 0 $f_{BAUD} \le \frac{f_{PER}}{8}$	$f_{BAUD} = \frac{f_{PER}}{8((2^{BSCALE} \cdot BSEL) + 1)}$	$BSEL = \frac{1}{2^{BSCALE}}\left(\frac{f_{PER}}{8 f_{BAUD}} - 1\right)$
Synchronous and SPI Master mode	$f_{BAUD} < \frac{f_{PER}}{2}$	$f_{BAUD} = \frac{f_{PER}}{2 \cdot (BSEL + 1)}$	$BSEL = \frac{f_{PER}}{2 f_{BAUD}} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

IMDL Software Series
USART

21.5 USART Initialization

USART initialization should use the following sequence:

1. Set the TxD pin value high, and optionally the XCK pin low.
2. Set the TxD and optionally the XCK pin as output.
3. Set the baud rate and frame format.
4. Set mode of operation (enables the XCK pin output in synchronous mode).
5. Enable the Transmitter or the Receiver depending on the usage.

For interrupt driven USART operation, global interrupts should be disabled during the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The transit and receive complete interrupt flags can be used to check that the Transmitter has completed all transfers, and that there are no unread data in the receive buffer.

IMDL Software Series

21.7 Data Reception - The USART Receiver

When the Receiver is enabled, the RxD pin is given the function as the Receiver's serial input. The direction of the pin must be set as input, which is the default pin setting.

21.7.1 Receiving Frames

The Receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the Receiver. When the first stop bit is received and a complete serial frame is present in the Receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The Receive Complete Interrupt Flag (RXCIF) is set, and the optional interrupt is generated.

The receiver buffer can be read by reading the Data Register (DATA) location. DATA should not be read unless the Receive Complete Interrupt Flag is set. When using frames with less than eight bits, the unused most significant bits are read as zero. If 9-bit characters are used, the ninth bit must be read from the RXB8 bit before the low byte of the character is read from DATA.

21.7.2 Receiver Error Flags

The USART Receiver has three error flags. The Frame Error (FERR), Buffer Overflow (BUFOVF) and Parity Error (PERR) flags are accessible from the Status Register. The error flags are located in the receive FIFO buffer together with their corresponding frame. Due to the buffering of the error flags, the Status Register must be read before the receive buffer (DATA), since reading the DATA location changes the FIFO buffer.

21.7.3 Parity Checker

When enabled, the Parity Checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit of the corresponding frame. If a parity error is detected the Parity Error flag is set.

21.7.4 Disabling the Receiver

A disabling of the Receiver will be immediate. The Receiver buffer will be flushed, and data from ongoing receptions will be lost.

21.7.5 Flushing the Receive Buffer

If the receive buffer has to be flushed during normal operation, read the DATA location until the Receive Complete Interrupt Flags is cleared.

IMDL Software Series

21.15.1 DATA - USART I/O Data Register

Bit	7	6	5	4	3	2	1	0
+0x00	RXB[[7:0]							
	TXB[[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register (DATA). The Transmit Data Buffer Register (TXB) will be the destination for data written to the DATA Register location. Reading the DATA Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the DREIF Flag in the STATUS Register is set. Data written to DATA when the DREIF Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. The data is then transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO and the corresponding flags in the Status Register (STATUS) will change state whenever the receive buffer is accessed (read). Always read STATUS before DATA in order to get the correct flags.

IMDL Software Series

21.15.2 STATUS - USART Status Register

Bit	7	6	5	4	3	2	1	0	
+0x01	RXCIF	TXCIF	DREIF	FERR	BUFOVF	PERR	-	RXB8	STATUS
Read/Write	R	R/W	R	R	R	R	R	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 - RXCIF: USART Receive Complete Interrupt Flag**

This flag is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). When the Receiver is disabled, the receive buffer will be flushed and consequently the RXCIF will become zero.

When interrupt-driven data reception is used, the receive complete interrupt routine must read the received data from DATA in order to clear the RXCIF. If not, a new interrupt will occur directly after the return from the current interrupt. This flag can also be cleared by writing a one to its bit location.

- **Bit 6 - TXCIF: USART Transmit Complete Interrupt Flag**

This flag is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data in the transmit buffer (DATA). The TXCIF is automatically cleared when the transmit complete interrupt vector is executed. The flag can also be cleared by writing a one to its bit location.

- **Bit 5 - DREIF: USART Data Register Empty Flag**

The DREIF indicates if the transmit buffer (DATA) is ready to receive new data. The flag is one when the transmit buffer is empty, and zero when the transmit buffer contains data to be transmitted that has not yet been moved into the Shift Register. DREIF is set after a reset to indicate that the Transmitter is ready. Always write this bit to zero when writing the STATUS register.

DREIF is cleared by writing DATA. When interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to DATA in order to clear DREIF or disable the Data Register Empty interrupt. If not, a new interrupt will occur directly after the return from the current interrupt.

IMDL Software Series

- **Bit 4 - FERR: Frame Error**

The FERR flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The bit is set if the received character had a Frame Error, i.e. when the first stop bit was zero, and cleared when the stop bit of the received data is one. This bit is valid until the receive buffer (DATA) is read. The FERR is not affected by setting the SBMODE bit in CTRLC since the Receiver ignores all, except for the first stop bit. Always write this bit location to zero when writing the STATUS register.

This flag is not used in Master SPI mode of operation.

- **Bit 3 - BUFOVF: Buffer Overflow**

The BUFOVF flag indicates data loss due to a receiver buffer full condition. This flag is set if a Buffer Overflow condition is detected. A Buffer Overflow occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This flag is valid until the receive buffer (DATA) is read. Always write this bit location to zero when writing the STATUS register.

This flag is not used in Master SPI mode of operation.

- **Bit 2 - PERR: Parity Error**

If parity checking is enabled and the next character in the receive buffer has a Parity Error this flag is set. If Parity Check is not enabled the PERR will always be read as zero. This bit is valid until the receive buffer (DATA) is read. Always write this bit location to zero when writing the STATUS register. For details on parity calculation refer to "Parity Bit Calculation" on page 241.

This flag is not used in Master SPI mode of operation.

- **Bit 1 - Reserved**

This bit is reserved and will always be read as zero. For compatibility with future devices, always write this bit to zero when this register is written.

- **Bit 0 - RXB8: Receive Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. When used, this bit must be read before reading the low bits from DATA.

This bit unused in Master SPI mode of operation.

IMDL Software Series

21.15.3 CTRLA – USART Control Register A

Bit	7	6	5	4	3	2	1	0	
+0x03	-	-	RXCINTLVL[1:0]		TXCINTLVL[1:0]		DREINTLVL[1:0]		CTRLA
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 - Reserved**

These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

- **Bit 5:4 - RXCINTLVL[1:0]: Receive Complete Interrupt Level**

These bits enable the Receive Complete Interrupt and select the interrupt level as described in Section 12. "Interrupts and Programmable Multi-level Interrupt Controller" on page 123. The enabled interrupt will be triggered when the RXCIF in the STATUS register is set.

- **Bit 3:2 - TXCINTLVL[1:0]: Transmit Complete Interrupt Level**

These bits enable the Transmit Complete Interrupt and select the interrupt level as described in Section 12. "Interrupts and Programmable Multi-level Interrupt Controller" on page 123. The enabled interrupt will be triggered when the TXCIF in the STATUS register is set.

- **Bit 1:0 - DREINTLVL[1:0]: USART Data Register Empty Interrupt Level**

These bits enable the Data Register Empty Interrupt and select the interrupt level as described in Section 12. "Interrupts and Programmable Multi-level Interrupt Controller" on page 123. The enabled interrupt will be triggered when the DREIF in the STATUS register is set.

IMDL Software Series

21.15.4 CTRLB - USART Control Register B

Bit	7	6	5	4	3	2	1	0	
+0x04	-	-	-	RXEN	TXEN	CLK2X	MPCM	TXB8	CTRLB
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:5 - Reserved**

These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

- **Bit 4 - RXEN: Receiver Enable**

Setting this bit enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FERR, BUFOVF, and PERR flags.

- **Bit 3 - TXEN: Transmitter Enable**

Setting this bit enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. Disabling the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxD port.

- **Bit 2 - CLK2X: Double Transmission Speed**

Setting this bit will reduce the divisor of the baud rate divider from16 to 8 effectively doubling the transfer rate for asynchronous communication modes. For synchronous operation this bit has no effect and should always be written to zero. This bit must be zero when the USART Communication Mode is configured to IRCOM.

This bit is unused in Master SPI mode of operation.

- **Bit 1 - MPCM: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, the USART Receiver ignores all the incoming frames that do not contain address information. The Transmitter is unaffected by the MPCM setting. For more detailed information see "Multi-processor Communication Mode" on page 248.

This bit is unused in Master SPI mode of operation.

- **Bit 0 - TXB8: Transmit Bit 8**

TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. When used this bit must be written before writing the low bits to DATA.

This bit is unused in Master SPI mode of operation.

IMDL Software Series

21.15.5 CTRLC - USART Control Register C

Bit	7	6	5	4	3	2	1	0
+0x05	CMODE[1:0]		PMODE[1:0]		SBMODE	CHSIZE[2:0]		
+0x05[1]	CMODE[1:0]		-	-	-	UDORD	UCPHA	-
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	1	1	0

Note: 1. Master SPI mode

- **Bits 7:6 - CMODE[1:0]: USART Communication Mode**

These bits select the mode of operation of the USART as shown in Table 21-6.

Table 21-6. CMODE bit settings

CMODE[1:0]	Group Configuration	Mode
00	ASYNCHRONOUS	Asynchronous USART
01	SYNCHRONOUS	Synchronous USART
10	IRCOM	IRCOM[1]
11	MSPI	Master SPI[2]

Notes: 1. See Section 22. "IRCOM - IR Communication Module" on page 256 for full description on using IRCOM mode.

- **Bits 5:4 - PMODE[1:0]: Parity Mode**

These bits enable and set the type of parity generation according to Table 21-7 on page 253. When enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the PMODE setting and if a mismatch is detected, the PERR flag in STATUS will be set.

These bits are unused in Master SPI mode of operation.

Table 21-7. PMODE Bits Settings

PMODE[1:0]	Group Configuration	Parity mode
00	DISABLED	Disabled
01		Reserved
10	EVEN	Enabled, Even Parity
11	ODD	Enabled, Odd Parity

IMDL Software Series

- **Bit 3 - SBMODE: Stop Bit Mode**

This bit selects the number of stop bits to be inserted by the Transmitter according to Table 21-8 on page 253. The Receiver ignores this setting.

This bit is unused in Master SPI mode of operation.

Table 21-8. SBMODE Bit Settings

SBMODE	Stop Bit(s)
0	1-bit
1	2-bit

- **Bit 2:0 - CHSIZE[2:0]: Character Size**

The CHSIZE[2:0] bits sets the number of data bits in a frame according to Table 21-9 on page 253. The Receiver and Transmitter use the same setting.

Table 21-9. CHSIZE Bits Settings

CHSIZE[2:0]	Group Configuration	Character size
000	5BIT	5-bit
001	6BIT	6-bit
010	7BIT	7-bit
011	8BIT	8-bit
100		Reserved
101		Reserved
110		Reserved
111	9BIT	9-bit

- **Bit 2 - UDORD: Data Order**

This bit sets the frame format. When written to one the LSB of the data word is transmitted first. When written to zero the MSB of the data word is transmitted first. The Receiver and Transmitter use the same setting. Changing the setting of UDORD will corrupt all ongoing communication for both receiver and transmitter.

- **Bit 1 - UCPHA: Clock Phase**

The UCPHA bit setting determine if data is sampled on the leading (first) edge or tailing (last) edge of XCKn. Refer to the "SPI Clock Generation" on page 239 for details.

IMDL Software Series

21.15.6 BAUDCTRLA - USART Baud Rate Register

Bit	7	6	5	4	3	2	1	0	
+0x06	BSEL[7:0]								BAUDCTRLA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 - BSEL[7:0]: USART Baud Rate Register**

This is a 12-bit value which contains the USART baud rate setting. The BAUDCTRLB contains the four most significant bits, and the BAUDCTRLA contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing BAUDCTRLA will trigger an immediate update of the baud rate prescaler.

21.15.7 BAUDCTRLB - USART Baud Rate Register

Bit	7	6	5	4	3	2	1	0	
+0x07	BSCALE[3:0]				BSEL[11:8]				BAUDCTRLB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:4 - BSCALE[3:0]: USART Baud Rate Scale factor**

These bits select the Baud Rate Generator scale factor. The scale factor is given in two's complement form from -7 (0b1001) to 7 (0b0111). The -8 (0b1000) setting is reserved. For positive scale values the Baud Rate Generator is prescaled by 2^{BSCALE}. For negative values the Baud Rate Generator will use fractional counting, which increases the resolution. See equations in Table 21-1 on page 238.

- **Bit 3:0 - BSEL[3:0]: USART Baud Rate Register**

This is a 12-bit value which contains the USART baud rate setting. The BAUDCTRLB contains the four most significant bits, and the BAUDCTRLA contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing BAUDCTRLA will trigger an immediate update of the baud rate prescaler.

IMDL Software Series
USART

21.16 Register Summary

21.16.1 Register Description - USART

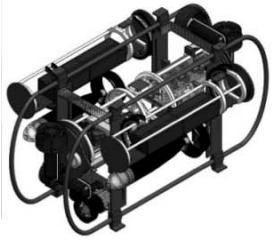
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	DATA	DATA[7:0]								249
+0x01	STATUS	RXCIF	TXCIF	DREIF	FERR	BUFOVF	PERR	-	RXB8	249
+0x02	Reserved	-	-	-	-	-	-	-	-	
+0x03	CTRLA	0	0	1	RXCINTLVL[1:0]	0	0	TXCINTLVL[1:0]	0	251
+0x04	CTRLB	0	0	0	RXEN	TXEN	CLK2X	MPCM	TX8	251
+0x05	CTRLC	0	CMODE[1:0]	0	PMODE[1:0]	0	SBMODE	0	CHSIZE[1:0]	253
+0x06	BAUDCTRLA	0	0	0	0	BSEL[7:0]		1	1	255
+0x07	BAUDCTRLB	1	0	BSCALE[3:0]		1	0	BSEL[11:8]	0	254

- Here is the register summary for the USART. We can see that serial communication requires the use of 7 registers
  - 1 for data
  - 1 for status
  - 3 for control
  - 2 for baud rate

21.17 Interrupt Vector Summary

Table 21-10. USART Interrupt vectors and their word offset address

Offset	Source	Interrupt Description
0x00	RXC_vect	USART Receive Complete Interrupt vector
0x02	DRE_vect	USART Data Register Empty Interrupt vector
0x04	TXC_vect	USART Transmit Complete Interrupt vector



AVR Studio 7 & Atmel ASF

- You can use GPIO or IOPORT functions to toggle I/O pins (& the external LEDs connected to them)
- The GPIO service requires ioport.h and most of the GPIO functions actually call the corresponding IOPORT functions in the ASF Library
- In the example I used the GPIO functions (the IOPORT functions are commented out) even though these functions have been deprecated.

See project DAQ_GPIO_USART

IMDL Software Series

Example below is for USART C0

usart.c - Notepad2 (Administrator)

File Edit View Settings ?

```
/*
 * usart.c
 *
 * Created: 1/22/2014 1:39:55 PM
 *  Author: Arroyo
 */
#include "usart.h"
#define BAUD_RATE_CONSTANT 0x88E     // constant for BAUD=57600  0x88E-2190  Source: Tim Martin

void usart_Init(void){               // USARTC0 Init
    PMIC_CTRL         |=   0x02;     // PMIC_MEDLVLEX_bm;
    PORTC_DIRSET       = 0x08;       // PC3 set to output
    USARTC0_CTRLB      = 0b00011100; // USART_RXEN_bm | USART_TXEN_bm | USART_CLK2X_bm;
    USARTC0_CTRLC      = 0b00000011; // USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIz
    USARTC0_BAUDCTRLA = 0x8E;        // Lower 8 bits of BAUD_RATE_CONSTANT
    USARTC0_BAUDCTRLB = 0xB8;        // 0xB0 | (uint16_t)(0x0FFF & BAUD_RATE_CONSTANT>>8);
    USARTC0_CTRLA     |= 0b00100000; // USART_RXCINTLVL_MED_gc;
}   // End USARTC0 Init

/*
ISR(USARTC0_RXC_vect){                              //AAA USART ISR {only needs to read USARTC0.DATA}
    uint8_t rcvData = 0;                            //AAA
    rcvData = USARTC0.DATA;                         //AAA
    if (rcvData == 'i') PORTR_OUTCLR = 0x02;        //AAA
    else if (rcvData == 'o') PORTR_OUTSET = 0x02;   //AAA
    else if (rcvData == 'j') PORTC_OUT = 0x01;      //AAA
    else if (rcvData == 'k') PORTC_OUT = 0x02;      //AAA
}
*/
```

IMDL Software Series

Example below is for USART D1

usart.c - Notepad2 (Administrator)

File Edit View Settings ?

```
/*
 * usart.c for the DAQ Board
 *
 * Created: 01/14/2015 2:59:55 PM
 *  Author: Arroyo
 */
#include "usart.h"
#define BAUD_RATE_CONSTANT 0x88E     // constant for BAUD=57600  0x88E-2190  Source: Tim Martin

void usart_Init(void){               // USARTD1 Init
    PMIC_CTRL         |=   0x02;     // PMIC_MEDLVLEX_bm;
    PORTD_DIRSET       = 0x02;       // PD1 set to output
    USARTD1_CTRLB      = 0b00011100; // USART_RXEN_bm | USART_TXEN_bm | USART_CLK2X_bm;
    USARTD1_CTRLC      = 0b00000011; // USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc;
    USARTD1_BAUDCTRLA = 0x8E;        // Lower 8 bits of BAUD_RATE_CONSTANT
    USARTD1_BAUDCTRLB = 0xB8;        // 0xB0 | (uint16_t)(0x0FFF & BAUD_RATE_CONSTANT>>8);
    USARTD1_CTRLA     |= 0b00100000; // USART_RXCINTLVL_MED_gc;
}   // End USARTD1 Init

```

Global replace USART D1 for USARTC0 & PORTD for PORTC

IMDL Software Series
USART

- The example uses the USART functions with the GPIO functions in the DAQ Board

See project DAQ_GPIO_USART

- If you wish to use Tim's USART functions, then import uart.c and uart.h from Tim's Software
- The ISR for the USART is in uart.c and it can be modified as needed.

See project DAQ_GPIO_USART
Change the include in main.c to <uart.h>

IMDL Software Series
Arduino USART

- The example uses the USART functions in the Arduino MEGA2560 Board to blink internal/ external LEDs and uses the serial port to display data and control the LEDs after blinking the set a few times. Serial input *i* turns on the internal LED on pin 13, and *o* turns the internal LED off. Similarly *j* & *k* alternate the two external LEDs on pins 12 & 11 {on, off} and {off, on} respectively

  - See Arduino Sketch: IMDL_Serial_External_LED

IMDL Software Series

```
/*
  EEL-5666 Example 2: Use of Digital Output Pins
  Turn on/off External LEDs connected to Pins 11 & 12
  alternatively and the internal LED on Pin 13 on/off.
  Add a delay in each state of 1/2 sec. Use serial input
  to control the LEDs. {i,o} for internal LED {j,k} for
  external LEDs.
  Arroyo January 18, 2016
*/

int extLED1 = 11;   // External LED connected to digital pin 11
int extLED2 = 12;   // External LED connected to digital pin 12
int intLED  = 13;   // Internal LED connected to digital pin 13

void setup( ) {
  pinMode(extLED1, OUTPUT);     // sets digital pin 12 as output
  pinMode(extLED2, OUTPUT);     // sets digital pin 12 as output
  pinMode(intLED,  OUTPUT);     // sets digital pin 13 as output
  Serial.begin(57600);         // connect to the serial port
  Serial.println("Arduino Serial External LEDs");}

  int rcvData=0, count=0;

void loop( ) {
  if (count < 1) {
    for (int j=0; j<5; j++){
      Serial.print("j= ");
      Serial.println(j);
      digitalWrite(13, HIGH);   // set the internal LED on
      digitalWrite(12, LOW);    // set the external LED1 off
      digitalWrite(11, HIGH);   // set the external LED2 on
      delay(500);               // wait for a 1/2 second
      digitalWrite(13, LOW);    // set the internal LED off
      digitalWrite(12, HIGH);   // set the external LED on
      digitalWrite(11, LOW);    // set the external LED off
      delay(500);               // wait for a 1/2 second
    }
```

Normal text file | length : 2201 lines : 57 | Ln : 1 Col : 1 Sel : 0 | 0

IMDL Software Series

```
38      digitalWrite(13, HIGH);   // set the internal LED  on
39      digitalWrite(12, HIGH);   // set the external LED1 on
40      digitalWrite(11, HIGH);   // set the external LED2 on
41    }
42    count=1;
43    // blink data only when you receive data:
44    if (Serial.available() > 0) {
45    // read the incoming byte:
46       rcvData = Serial.read();
47    // say what you got:
48       Serial.print("I received: ");
49       Serial.write(rcvData);
50       Serial.println();
51       if (rcvData == 'i') digitalWrite(13, HIGH);      // internal LED on
52       else if (rcvData == 'o') digitalWrite(13, LOW);  // internal LED off
53       else if (rcvData == 'j') {digitalWrite(12,LOW); digitalWrite(11,HIGH);}
54       else if (rcvData == 'k') {digitalWrite(12,HIGH); digitalWrite(11,LOW);}
55       }
56  }
57
```

Normal text file | length : 2201 lines : 57 | Ln : 5 Col : 1 Sel : 55 | 2 | UNIX

The End!