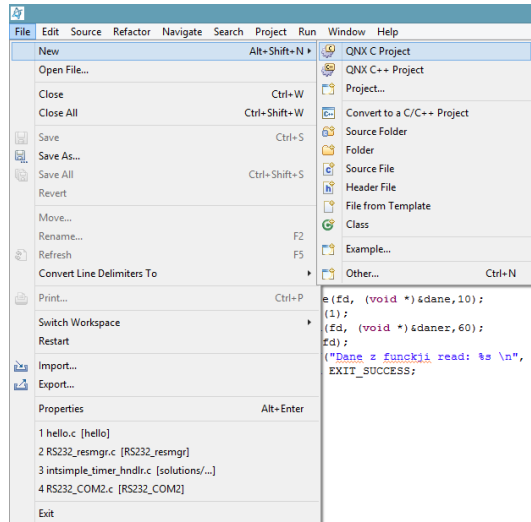
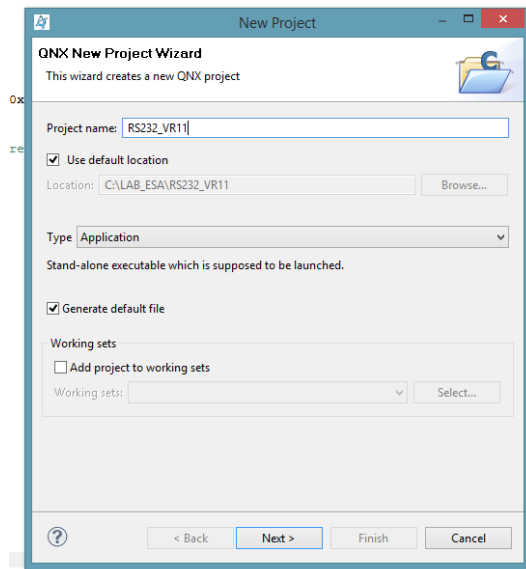


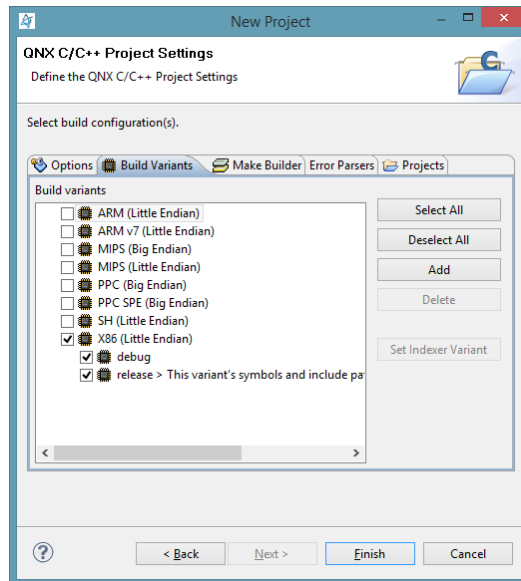
## 1. Tworzenie nowego projektu.



Wybieramy opcję z menu głównego New->QNX C Project.

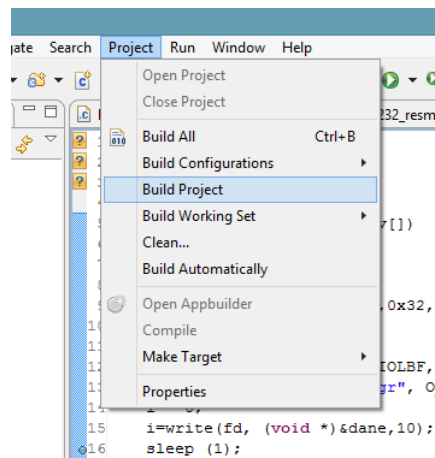


Wprowadzamy nazwę przechodzimy do następnego kroku NEXT.



Wybieramy platformę docelową oraz warianty kompilacji. FINISH kończy proces tworzenia projektu.

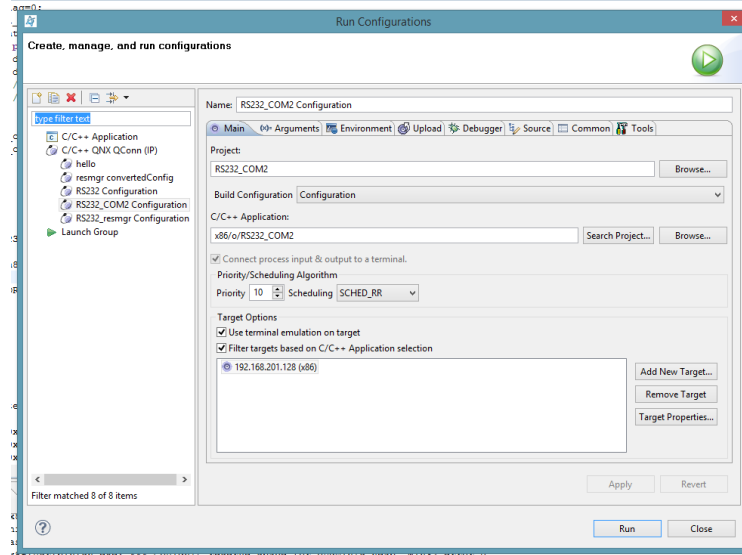
## 2. Kompilacja projektu.



Wybieramy z menu głównego opcję Project->Build Project. Wynik kompilacji pojawi się w zakładce Console. W zakładce Problem wyświetlone zostaną ewentualne błędy i problemy.

### 3. Uruchomienie aplikacji

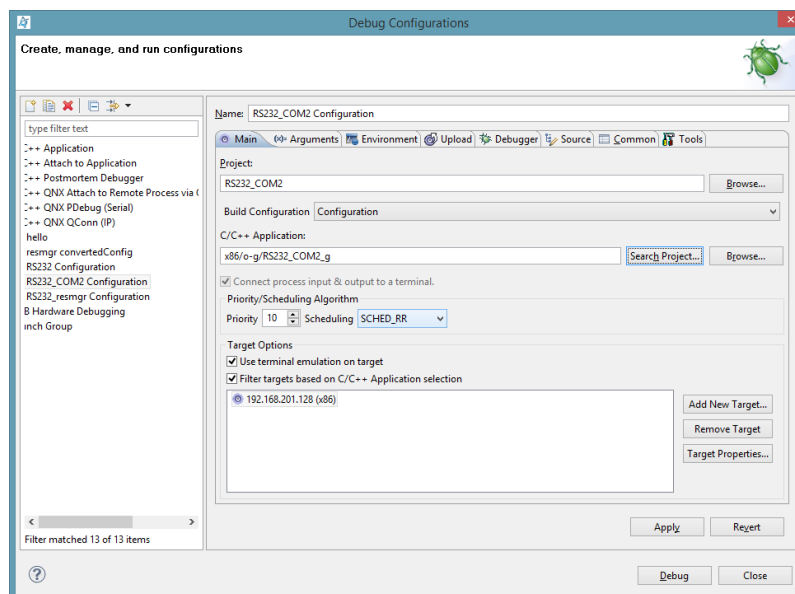
Z menu głównego wybieramy opcję za pierwszym razem Run->Run Configuration, w kolejnych Run->Run



W polu po lewej stronie tworzymy nową konfigurację uruchamiania C/C++ QNX QConn IP. W polu Project Browse wybieramy projekt z przestrzeni roboczej. W polu C/C++ Application „Search Project” wybieramy aplikację do uruchomienia. W zakładce „Arguments” możemy prowadzić argumenty wejściowe aplikacji. „Run” uruchamia aplikację na platformie docelowej. W polu Target Option wybieramy odpowiednią platformę docelową.

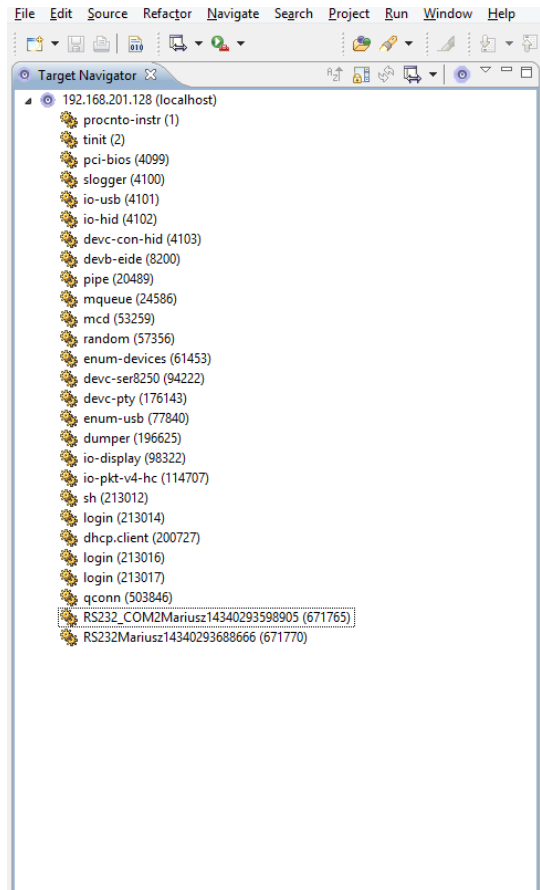
### 4. Debugowanie aplikacji.

Podobnie jak w pkt 3. używamy opcji Run->Debug Configuration.



## 5. Perspektywa QNX System Information

Uruchomione programy powinny znaleźć się na liście procesów uruchomionych na platformie docelowej w oknie Target Navigator Perspektywy QNX System Information .



## 6. Programowanie interfejsu UART na platformie x86.

Interfejs szeregowy w komputerze PC posiada następujące rejestry:

**UART register to port conversion table**

I/O port	DLAB = 0		DLAB = 1	
	Read	Write	Read	Write
base	<b>RBR</b> receiver buffer	<b>THR</b> transmitter holding	<b>DLL</b> divisor latch LSB	
base + 1	<b>IER</b> interrupt enable	<b>IER</b> interrupt enable	<b>DLM</b> divisor latch MSB	
base + 2	<b>IIR</b> interrupt identification	<b>FCR</b> FIFO control	<b>IIR</b> interrupt identification	<b>FCR</b> FIFO control
base + 3	<b>LCR</b> line control			
base + 4	<b>MCR</b> modem control			
base + 5	<b>LSR</b> line status	– factory test	<b>LSR</b> line status	– factory test
base + 6	<b>MSR</b> modem status	– not used	<b>MSR</b> modem status	– not used
base + 7	<b>SCR</b> scratch			

**Ilustracja ze strony: <http://www.lammertbies.nl/comm/info/serial-uart.html>**

Niektóre rejestry pełnią zróżnicowane role w zależności od wykonywanej na nich operacji zapis/odczyt. Znaczenie pewnych rejestrów zależy od ustawień innych rejestrów, np. rejestry spod adresu bazowego i adresu bazowego+1 mają różne znaczenie w zależności od wartości bitu DLAB w rejestrze LCR.

Programowanie interfejsu szeregowego sprowadza się do wykonania następujących kroków:

- ❖ Zainicjowanie urządzenia poprzez wpisanie odpowiednich wartości do jego rejestrów.
- ❖ Wyłączenie przerw w rejestrze IER:  
Wpisanie do rejestru IER – 0x00;

its in the IER, interrupt enable register. A bit value 1 indica

**IER : Interrupt enable register**

Bit	Description
0	Received data available
1	Transmitter holding register empty
2	Receiver line status register change
3	Modem status register change
4	Sleep mode (16750 only)
5	Low power mode (16750 only)
6	reserved
7	reserved

**Ilustracja ze strony: <http://www.lammertbies.nl/comm/info/serial-uart.html>**

- ❖ Ustawienie prędkości transmisji:
  - ❖ Bit DLAB = 1 w rejestrze LCR;
  - ❖ Wpisanie odpowiednich wartości do rejestrów DLL – Divisor Latch LSB;
  - ❖ Wpisanie odpowiednich wartości do rejestrów DLM – Divisor Latch MSB;

- ❖ Ustawienie parametrów transmisji, rejestr LCR:  
Wpisanie odpowiednich wartości w rejestrze LCR – bit DLAB = 0;

**LCR : line control register**

Bit	Value		Description	
<b>0,1</b>	<b>Bit 1</b>	<b>Bit 0</b>	Data word length	
	0	0	5 bits	
	0	1	6 bits	
	1	0	7 bits	
	1	1	8 bits	
<b>2</b>	0		1 stop bit	
	1		1.5 stop bits (5 bits word) 2 stop bits (6, 7 or 8 bits word)	
<b>3,4,5</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	
	x	x	0	No parity
	0	0	1	Odd parity
	0	1	1	Even parity
	1	0	1	High parity (stick)
	1	1	1	Low parity (stick)
<b>6</b>	0		Break signal disabled	
	1		Break signal enabled	
<b>7</b>	0		DLAB : RBR, THR and IER accessible	
	1		DLAB : DLL and DLM accessible	

**Ilustracja ze strony: <http://www.lammertbies.nl/comm/info/serial-uart.html>**

- ❖ Ustawienie odpowiednich parametrów wewnętrznej pamięci FIFO, wpisanie odpowiedniej wartości do rejestru FCR:

**FCR : FIFO control register**

Bit	Value		Description
<b>0</b>	0		Disable FIFO's
	1		Enable FIFO's
<b>1</b>	0		-
	1		Clear receive FIFO
<b>2</b>	0		-
	1		Clear transmit FIFO
<b>3</b>	0		Select DMA mode 0
	1		Select DMA mode 1
<b>4</b>	0		Reserved
<b>5</b>	0		Reserved (8250, 16450, 16550)
	1		Enable 64 byte FIFO (16750)
<b>6,7</b>	<b>Bit 7</b>	<b>Bit 6</b>	Receive FIFO interrupt trigger level
	0	0	1 byte
	0	1	4 bytes
	1	0	8 bytes
	1	1	14 bytes

**Ilustracja ze strony: <http://www.lammertbies.nl/comm/info/serial-uart.html>**

- ❖ Włączenie odpowiednich wyjść i sygnałów kontroli transmisji w rejestrze MCR:

**MCR : Modem control register**

Bit	Description
<b>0</b>	Data terminal ready
<b>1</b>	Request to send
<b>2</b>	Auxiliary output 1
<b>3</b>	Auxiliary output 2
<b>4</b>	Loopback mode
<b>5</b>	Autoflow control (16750 only)
<b>6</b>	Reserved
<b>7</b>	Reserved

**Ilustracja ze strony: <http://www.lammertbies.nl/comm/info/serial-uart.html>**

- ❖ Napisanie funkcji obsługi przerwań od układu UART. Źródła przerwań od układu transmisji szeregowej można rozróżnić sprawdzając zawartość rejestru IIR:

IIR : Interrupt identification register					
Bit	Value			Description	Reset by
0	0			Interrupt pending	-
	1			No interrupt pending	-
1,2,3	Bit 3	Bit 2	Bit 1	Modem status change	MSR read
	0	0	0	Transmitter holding register empty	IIR read or THR write
	0	0	1	Received data available	RBR read
	0	1	0	Line status change	LSR read
	0	1	1	Character timeout (16550)	RBR read
4	0			Reserved	-
5	0			Reserved (8250, 16450, 16550)	-
	1			64 byte FIFO enabled (16750)	-
6,7	Bit 7	Bit 6		No FIFO	-
	0	0		Unusable FIFO (16550 only)	-
	1	0		FIFO enabled	-
	1	1			

**Ilustracja ze strony: <http://www.lammertbies.nl/comm/info/serial-uart.html>**

Zawartość bitów 1 – 3 rejestru IIR mówi nam, z jakim przerwaniem mamy do czynienia. „1” logiczna na bicie 1 mówi nam, że bufor nadajnika jest pusty i można nadać kolejny znak. „1” logiczna na bicie 2 mówi nam że w buforze odbiornika jest znak gotowy do odbioru. Sprawdzanie bitów rejestru IIR jest konieczne gdyż z punktu widzenia procesora generowane jest jedno przerwanie dotyczące różnych zdarzeń.

- ❖ Włączenie przerwania w układzie UART poprzez ustawienie odpowiednich bitów w rejestrze IER.

## 7. Obsługa przerwania w QNX Neutrino

W QNX Neutrino możemy obsługiwać przerwanie sprzętowe w dwojaki sposób poprzez:

- ❖ napisanie własnej procedury obsługi przerwania, która uruchamia odpowiednie zdarzenie w systemie;
- ❖ obsługę w wątku zdarzenia wygenerowanego przez jądro.

Na laboratorium wykorzystamy pierwszy sposób obsługi przerwania. W tym celu musimy wykonać następujące kroki:

- ❖ Zdefiniować odpowiednią strukturę odpowiedzialną za zdarzenie generowane po wystąpieniu przerwania:

```
struct sigevent int_event; // the event to wake up the thread;
```

- ❖ Napisać funkcję obsługi przerwania:

```
const struct sigevent *hdlr( void *blah, int id ){
    static int st = 0;
    st = in8(PORT+2); //sprawdzenie źródła przerwania
    if (przerwanie od odbiornika){
        //wykonaj operacje
        return &int_event; // zgłoś zdarzenie
    }
    if (przerwanie od nadajnika){
        //wykonaj operacje
        return &int_event; // zgłoś zdarzenie
    }
}
```

```
}
```

```
return NULL; //w przeciwnym wypadku zwróć NULL
```

```
}
```

❖ W pętli głównej programu:

❖ włącz obsługę operacji I/O:

```
// request I/O privity  
ThreadCtl(_NTO_TCTL_IO, 0 );
```

❖ Zainicjuj structure zdarzenia:

```
SIGEV_INTR_INIT( &int_event );
```

❖ Powiąż własnego handler z przerwaniem:

```
id = InterruptAttach(IRQn, handler, NULL, 0,  
_NTO_INTR_FLAGS_TRK_MSK );
```

❖ Włącz przerwania w urządzeniu – odpowiedni wpis w rejestrze IER.

np. out8(PORT+1, 0x01);

❖ W nieskończonej pętli while oczekuj na przerwanie i odpowiednio zareaguj:

```
while (1) {  
    // block here waiting for the event  
    InterruptWait(0, NULL );  
    // po wystąpieniu przerwania wykonywaj odpowiedni kod  
    // if using a high frequency interrupt, don't print every interrupt  
    printf ("%s: we got an event after 1500 timer ticks and unblocked\n",  
prognam);  
}
```