

Praktikum Systemprogrammierung

Versuch 0 - Vorkurs

► Inhalt des Vorkurses

- Projekterstellung in AtmelStudio
- Kennenlernen der Hardware
- Grundlagen der Programmiersprache C
- Ansprechen der Hardware im Programmcode

Benötigte Unterlagen

- ▶ Begleitendes Dokument
 - Beantwortet fast alle Fragen zum Praktikum
 - Zu finden im Moodle Lernraum
- ▶ Zu finden auf Netzlaufwerk
 - <\\\\pcpool.rz.rwth-aachen.de\\files\\group\\embedded\\psp\\>
 - Vom RWTH-Netz aus erreichbar

Zu finden unter `Computer → GruppeXX → public:`

- ▶ Versuchsunterlagen zu V0
- ▶ Diese Präsentation

Die Programmiersprache C

- ▶ Eignet sich zur Verwendung auf Mikrokontrollern
- ▶ Der Programmierer muss sich um vieles selbst kümmern
- ▶ Ausführliche Einführung in C befindet sich im begleitenden Dokument

Binär- und Hexadezimalzahlen

- ▶ Zahlen können in C als Dezimal-, Binär- oder Hexadezimalzahlen angegeben werden

```
// Initialisierung von var1 (dezimal)
uint8_t var1 = 200;
```

```
// Initialisierung von var2 (binär durch Voranstellen von
0b)
uint8_t var2 = 0b11001000; // var2 = 200;
```

```
// Initialisierung von var3 (hex durch Voranstellen von 0x)
uint8_t var3 = 0xC8; // var3 = 200;
```

Bitweise Operatoren auf Zahlen

- ▶ Einzelne Bits von Ausdrücken lassen sich durch bitweise Operatoren manipulieren
- ▶ Die Operanden werden Bit für Bit durchgegangen und immer zwei Bits zu einem dritten Bit verknüpft
- ▶ Bsp.: Veroderung

```
// Erstellung zweier Variablen mit jeweils 8 Bit
uint8_t var1 = 0b10101010;
uint8_t var2 = 0b00000111;
// Verodert: 0b10101111;

uint8_t var3 = var1 | var2;
```

Funktion bitweiser Operatoren

▶ Veroderung: Bits setzen (auf 1 setzen)

```
// Bit 2 (beginnend bei 0) von var setzen  
var |= 0b00000100; // kurz für: var = var | 0b00000100;
```

▶ Verundung: Bits löschen (auf 0 setzen)

```
// Bit 2 von var löschen  
var &= 0b11111011;
```

Konstanten im Quellcode, die dazu bestimmt sind, immer die gleichen Bits einer Variable zu manipulieren, werden „Bitmasken“ genannt

▶ XOR: Bits invertieren

```
// Bit 2 von var invertieren  
var ^= 0b00000100;
```

Mehr Operatoren...

► Bit-Shifting

- Verschiebt Bits eines Wertes
- Auf der Seite, in die geschoben wird, fallen Bits raus
- Auf der anderen Seite rücken Nullen nach

```
uint8_t var = 0b00111111;  
// Inhalt von var um zwei Stellen nach links schieben  
uint8_t l = var << 2; // 0b11111100  
// Und nach rechts...  
uint8_t r = var >> 2; // 0b00001111
```

► Bitmasken lassen sich einfach durch Bit-Shifting erzeugen

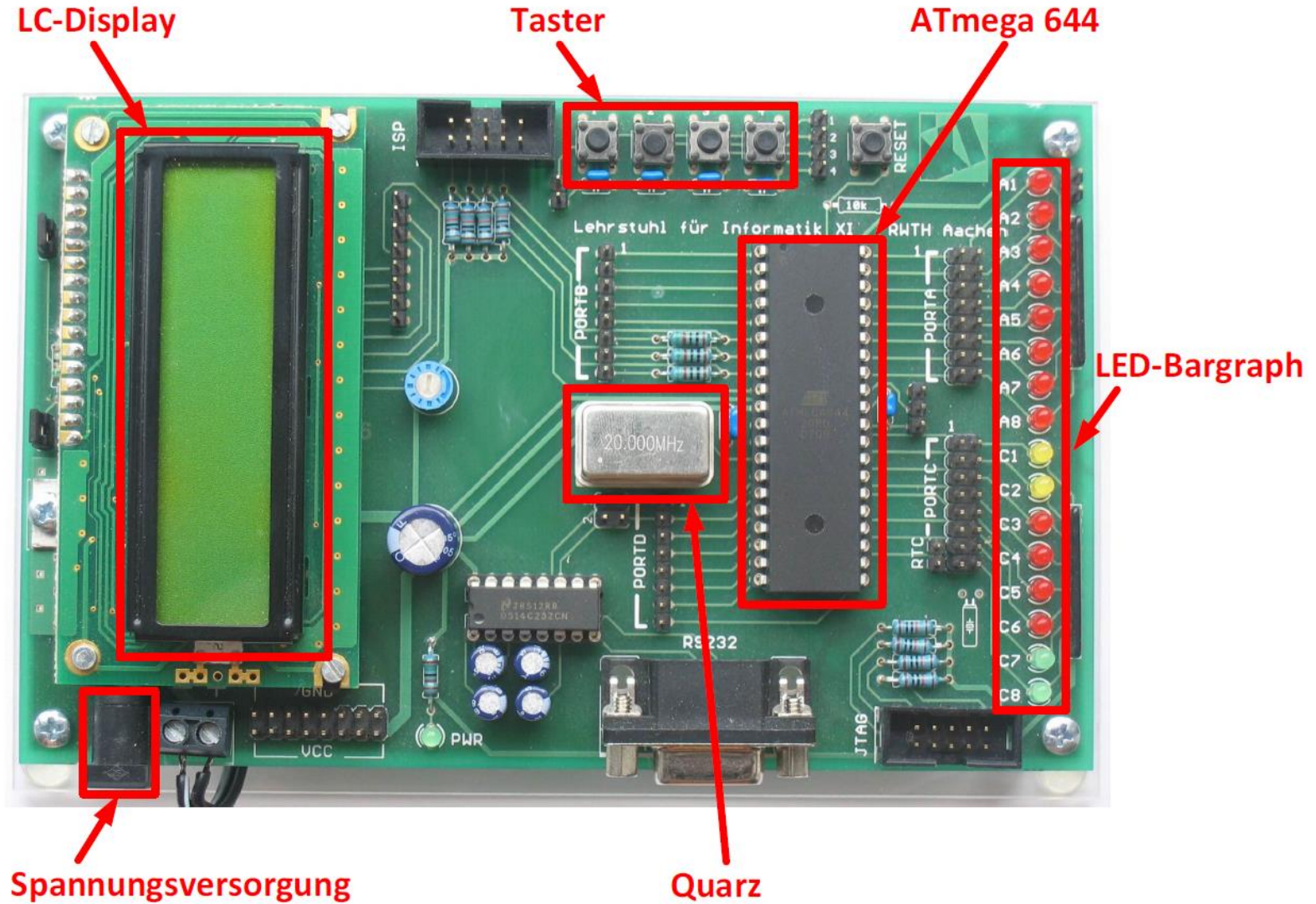
```
uint8_t mask1 = (1 << 5); // 0b00100000  
uint8_t mask2 = (1 << 5) | (1 << 3); // 0b00101000
```


► Bitweise Negation

- Invertiert jedes Bit

```
uint8_t mask1 = ~(1 << 5); // 0b11011111
uint8_t mask2 = ~((1 << 5) | (1 << 3)); // 0b11010111
```

Mikrocontroller



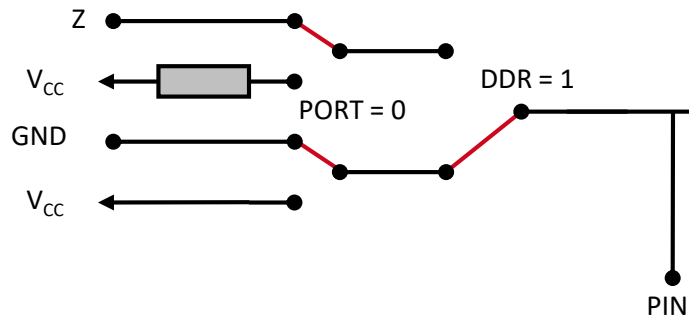
- ▶ ATmega644 verfügt über 4 Ports (A, B, C, D)
 - Jeder Port besitzt 8 Pins
- ▶ Jedem Port sind drei Kontrollregister zugeordnet:
 - $PORT_x$ ($x \in \{A, B, C, D\}$)
 - PIN_x
 - DDR_x
- ▶ Jedes Register umfasst 8 Bits
 - Jedes Bit ist einem Pin des Ports zugeordnet
- ▶ Können direkt im Code angesprochen werden

Kontrollregister für Ports

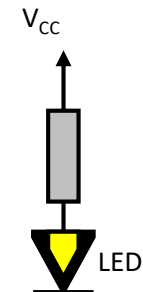
- ▶ Data Direction Register (DDR):
 - Lesen/schreiben
 - Jedes Bit legt fest, ob der entsprechende Pin Ein- oder Ausgang ist
- ▶ Port Register (PORT):
 - Lesen/schreiben
 - Legt für Ausgangspins (siehe DDR) fest, ob der Wert *high* oder *low* ist
 - Legt für Eingangspins fest, ob er mit einem *pull-up* Widerstand verbunden ist
- ▶ Port Input Register (PIN):
 - Nur lesen
 - Enthält den aktuellen Wert eines Pins (Ein- und Ausgang)

Beispiel: LED

μC



Normalerweise verbunden mit V_{CC} !

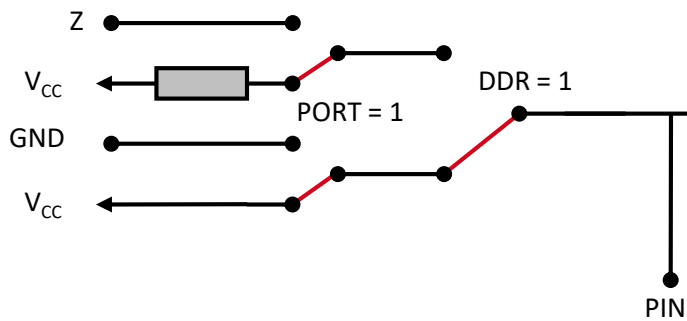


LED ist an!

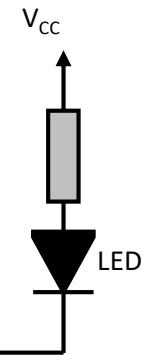
DDR	PORT	Signal an PIN
0	0	1
0	1	1
1	0	0
1	1	1

Beispiel: LED

μC



Normalerweise verbunden
mit V_{CC} !



LED ist aus!

DDR	PORT	Signal an PIN
0	0	1
0	1	1
1	0	0
1	1	1

Beispiel: LED

▶ Beispiel

```
// 1. Pin D1 als Ausgang konfigurieren (Bit 1 von DDRD auf 1 setzen)
```

```
DDRD |= 0b00000010;
```

```
// 2. LED einschalten (Bit 1 von PORTD auf 0 setzen)
```

```
PORTD &= 0b11111101;
```

```
// 3. LED wieder ausschalten (Bit 1 von PORTD auf 1 setzen)
```

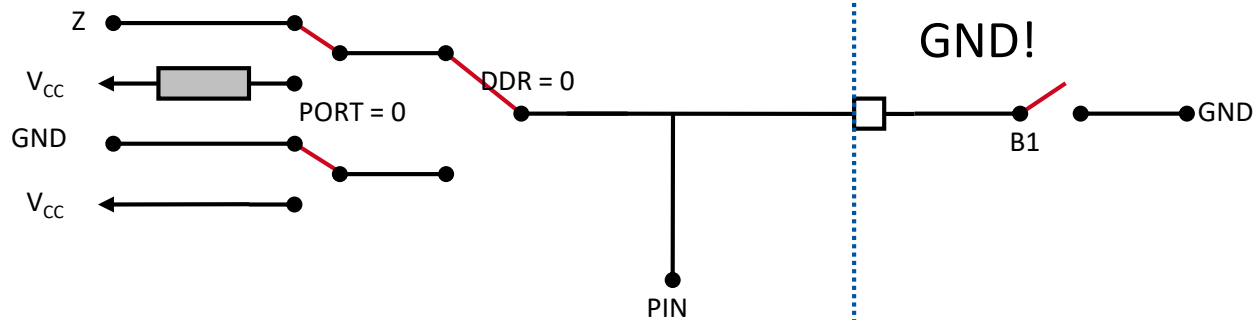
```
PORTD |= 0b00000010;
```

Buttons

- ▶ Können verwendet werden um Eingaben am Controller zu tätigen
- ▶ Angeschlossen an die Pins des Mikrocontrollers
 - Müssen als Eingang konfiguriert werden
- ▶ Wird Button gedrückt liegt eine 0 am Mikrocontroller an

Beispiel: Button

μC



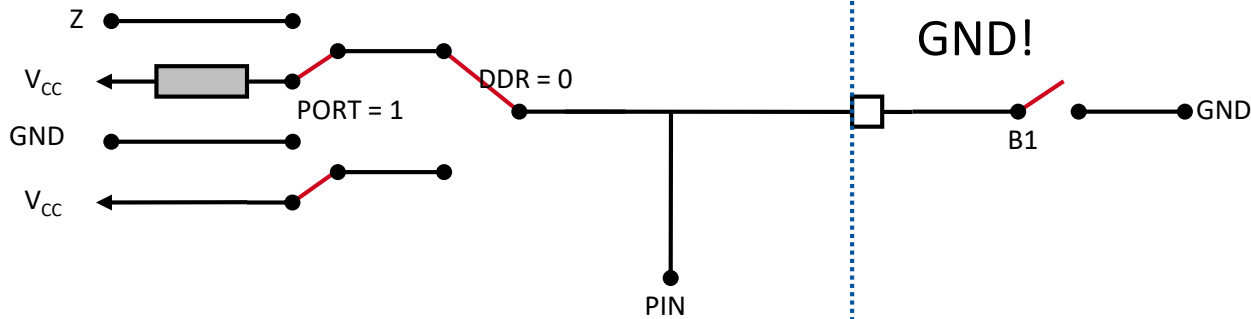
**PIN hat kein
definiertes Level
wenn B1 ist offen**

Normalerweise verbunden mit
GND!

DDR	PORT	Signal an PIN
0	0	?
0	1	1
1	0	0
1	1	1

Beispiel: Button

μC



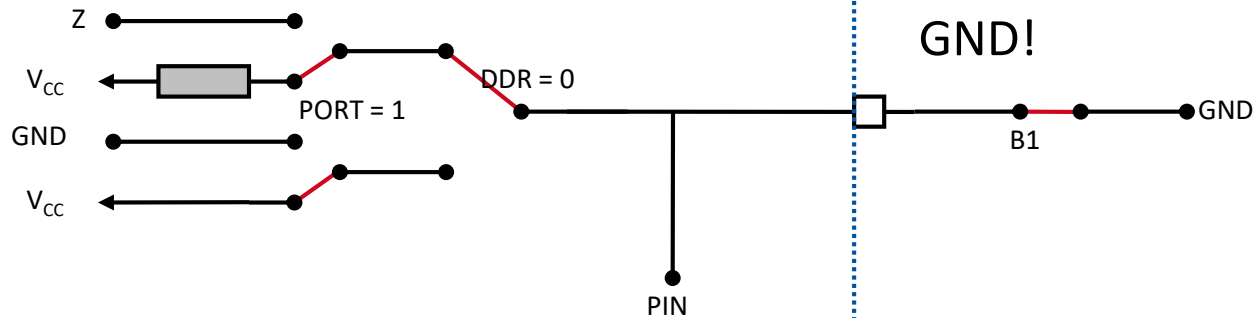
**Pull-up
Widerstand löst
das Problem**

Normalerweise verbunden mit
GND!

DDR	PORT	Signal an PIN
0	0	?
0	1	1
1	0	0
1	1	1

Beispiel: Button

μC



**Die Spannung kann
von B1 auf 0V
gezogen werden**

Normalerweise verbunden mit
GND!

DDR	PORT	Signal an PIN
0	0	0
0	1	0
1	0	0
1	1	⚡

Beispiel: Button

▶ Beispiel

```
//1. Pin D1 als Eingang konfigurieren (Bit 1 von DDRD auf 0 setzen)
```

```
DDRD &= 0b11111101;
```

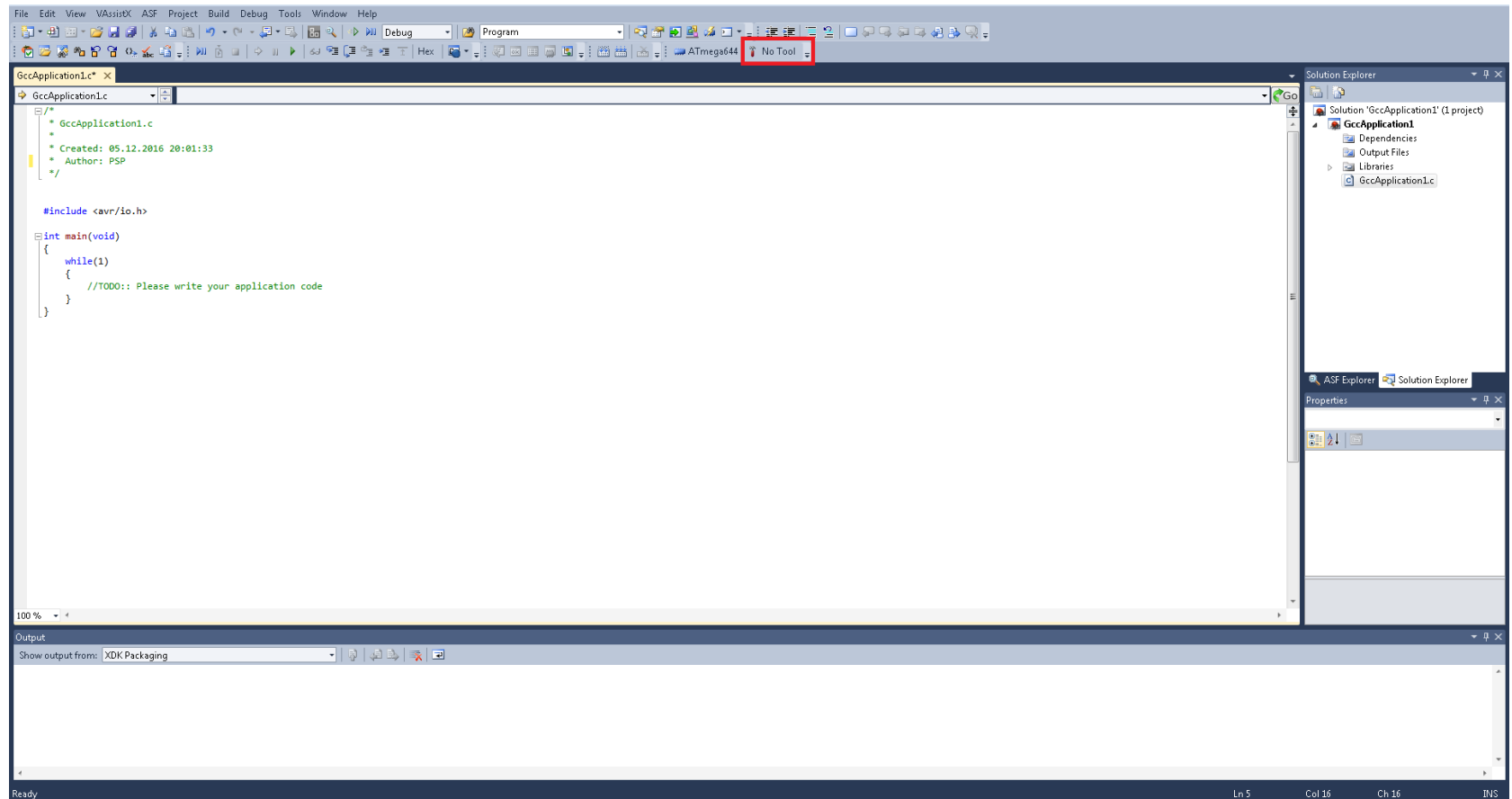
```
// 2. Pullup-Widerstand an Pin D1 aktivieren (Bit 1 von PORTD auf 1 setzen)
```

```
PORTD |= 0b00000010;
```

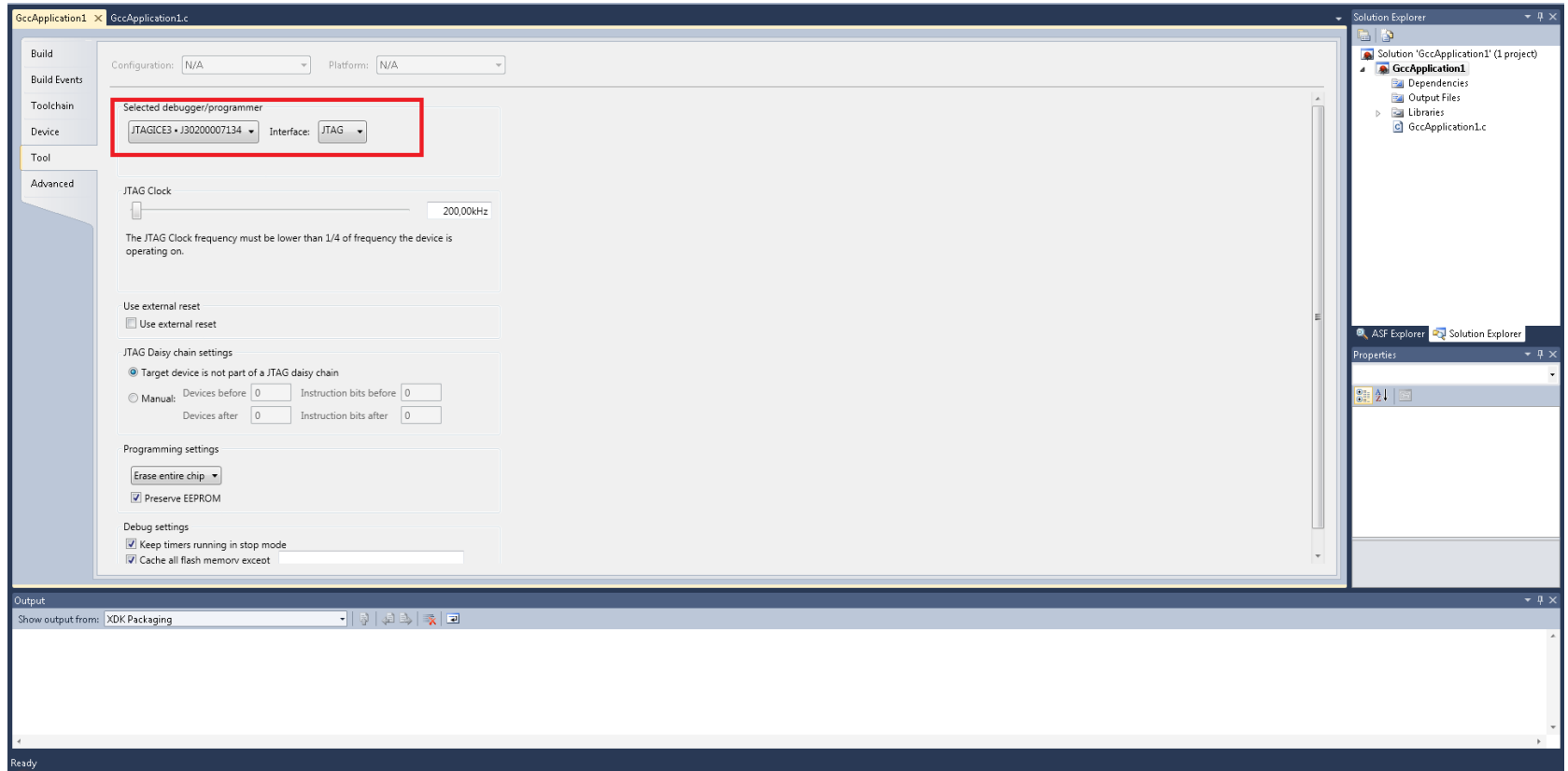
```
// 3. Pin D1 auslesen (Bit extrahieren und entsprechend verschieben)
```

```
uint8_t pinState = (PIND & 0b00000010) >> 1;
```

Projekterstellung



Projekterstellung



Weiterer Ablauf

- ▶ Nun seid ihr dran
- ▶ PDF mit heutigen Aufgaben im *public*-Ordner
- ▶ Bei Fragen einfach melden
 - Wir beißen nicht 😊



Vielen Dank für eure Aufmerksamkeit