arduino und esp32

Mit einem Source code

engelbert gruber

24.11.2020
Rev. 0.1

INTRODUCTION

Wir nehmen das Programm von “arduino begin” den logic analyser.

```
// logic analyzer

// Kanalnamen und Pins
char Names[] = "Eins Zwei Drei Vier";
int Pin[4] = { A0, A1, A2, A3 };
int OutputPin = 8;
void setup() {
  Serial.begin(9600);
  Serial.println(Names);
  pinMode(OutputPin, OUTPUT);
}

void print_zeitraster() {
  static int ms = 0;
  int t = millis();
```

```
  // Zeitraster: bei jedem 1000-er eine Spitze
  if ((t - ms) > 1000) {
    Serial.print(10);
    ms = t;
  }
  else {
    Serial.print(0);
  }
  Serial.print(" ");
}

bool pause() {
  static bool _pause = false;
  // eine Eingabe schaltet von Pause auf Run und umgekehrt.
  if (Serial.available()) {
    Serial.readStringUntil(10); // die Zeile bis zum Linefeed
lesen
    _pause = ! _pause;
  }
  return _pause;
}

void loop() {
  static int loopcnt = 0;
  loopcnt++;
  digitalWrite(OutputPin, (loopcnt / 10) % 2);
  if (pause()) {
    return; // nichts ausgeben damit der Graph stehen bleibt
  }
  print_zeitraster();
  // Kanaele lesen und ausgeben
  for (int i = 0; i < 4; i++) {
    Serial.print(digitalRead(Pin[i]) + (i + 1) * 2);
    Serial.print(" ");
  }
  Serial.println("");
}
```

Und versuchen es am esp32 zum Laufen zu bringen. Dazu brauchen wir (vielleicht) nur die Pins zu ändern.

MATERIALS

1. Esp32 und arduino Uno
2. PC mit arduino-IDE
3. Interesse

Versuch

Wir ersetzen die Pin Deklaration.

```
int Pin[4] = { A0, A1, A2, A3 };
```

durch

```
int Pin[4] = { 12, 14, 27, 26 };
```

Beim kompilieren bekommen wir eine Fehlermeldung weil irgendwo schon eine Funktion “pause” deklariert ist. Nach dem Umbenennen in “pause_printing” kompiliert die Datei und wird zum esp32 übertragen.

IT WORKS … Glück gehabt, dass der Pin 8 den ich als Ausgang verwende am esp32 nichts

tut.

C++

Wenn die IDE aus der arduino-Codedatei ein Programm erzeugt passiert, unter der Annahme einer normalen C++-Architektur , wahrscheinlich folgendes.

1. Die Datei wird durch den Präprozessor geschickt, dieser führt die Anweisungen in den Zeilen die mit einem Hash Zeichen "#" beginnen aus.
2. Die daraus resultierende Datei wird durch den COmpiler geschickt. Dieser erzeugt ein Object file (ACHTUNG: das hat NICHTS mit objekt orientierten Programmieren zu tun).
3. Der Linker verbindet die Objekt Datei(en) mit den benötigten Bibliotheken und der Laufzeitumgebung und erzeugt ein ausführbares Programm.
4. Diese Programm wird dann zum arduino übertragen und dort ausgeführt.

Interessant für uns ist ob wir mit dem Präprozessor weil wir dort vielleicht unterscheiden könne ob das Programm für eine arduino/avr oder esp kompiliert wird.

PRÄPROZESSOR

Der Präprozessor schreibt Sourcecode um :

```
#define A 9
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(A);
  delay(500);
}
```

Am Serial Monitor word "9" ausgegeben. Der Präprozessor ersetzt das A durch 9.

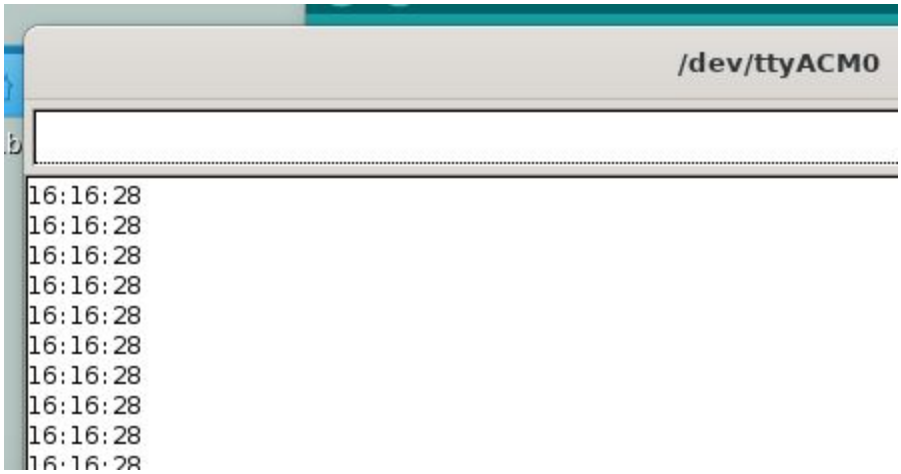
Wenn man "`Serial.println(__LINE__);`" schreibt gibt obiges Programm "7" aus, weil die Anweisung "`Serial.println(__LINE__);`" in der 7-ten Zeile steht.

Der Präprozessor hat einige eingebaute Variablen: google "preprocessor variables" :

https://www.cs.uic.edu/~jbell/CourseNotes/C_Programming/Preprocessor.html

Listet als "predfeinde macros" : __LINE__, __FILE__, __DATE__, __TIME__ ... man kann also ins Programm hineinschreiben, wann es kompiliert wurde.

```
Serial.println(__TIME__);
```



Die "16:16:28" sind der Zeitpunkt an dem das Programm erstellt wurde, beziehungsweise, zu dem der Präprozessor ausgeführt wurde.

Wir müssten aber die Prozessorarchitektur abprüfen, dann eben https://en.wikipedia.org/wiki/C_preprocessor. Von dort weiter nach https://sourceforge.net/p/predef/wiki/Home/ . Leider nichts richtiges gefunden ... ausser einem Notnagel "`F_CPU`", die CPU-Frequenz "`Serial.println(F_CPU);`" gibt beim arduino "16000000" aus beim esp32 "240000000".

```
// logic analyzer

// Kanalnamen und Pins
char Names[] = "Zeit Eins Zwei Drei Vier";
#if F_CPU == 240000000
char ARCH[] = "esp32 240MHz";
int Pin[4] = { 12, 14, 27, 26 };
int OutputPin = 33;
#else
char ARCH[] = "arduino 16MHz";
int Pin[4] = { A0, A1, A2, A3 };
int OutputPin = 8;
```

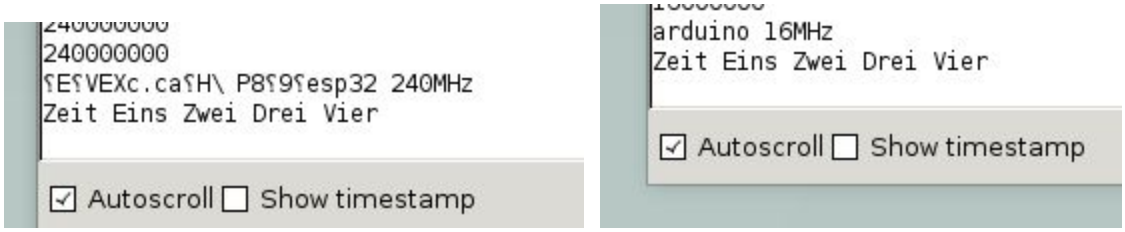
```
#endif

void setup() {
  Serial.begin(9600);
  Serial.println(ARCH);
  Serial.println(Names);
  pinMode(OutputPin, OUTPUT);
}
```

Die Funktion pause wird umbenannt und der Wert für “pause_” ist mit “true” initialisiert,

```
bool pause_printing() {
  static bool _pause = true;
  // eine Eingabe schaltet von Pause auf Run und umgekehrt.
  if (Serial.available()) {
    Serial.readStringUntil(10); // die Zeile bis zum Linefeed lesen
    _pause = ! _pause;
  }
  return _pause;
}
```

dann kann man das Programm starten und am Serial Monitor schauen was am Anfang ausgegeben wird.



Wir ändern wieder auf “`_pause = false;`”.

DAS PROGRAMM

```
// logic analyzer

// Kanalnamen und Pins
char Names[] = "Zeit Eins Zwei Drei Vier";
#if F_CPU == 240000000
```

```
char ARCH[] = "esp32 240MHz";
int Pin[4] = { 12, 14, 27, 26 };
int OutputPin = 33;
#else
char ARCH[] = "arduino 16MHz";
int Pin[4] = { A0, A1, A2, A3 };
int OutputPin = 8;
#endif

void setup() {
  Serial.begin(9600);
  Serial.println(ARCH);
  Serial.println(Names);
  pinMode(OutputPin, OUTPUT);
}

void print_zeitraster() {
  static int ms = 0;
  int t = millis();
  // Zeitraster: bei jedem 1000-er eine Spitze
  if ((t - ms) > 1000) {
    Serial.print(10);
    ms = t;
  }
  else {
    Serial.print(0);
  }
  Serial.print(" ");
}

bool pause_printing() {
  static bool _pause = false;
  // eine Eingabe schaltet von Pause auf Run und umgekehrt.
  if (Serial.available()) {
    Serial.readStringUntil(10); // die Zeile bis zum Linefeed lesen
    _pause = ! _pause;
  }
  return _pause;
}

void loop() {
```

```
  static int loopcnt = 0;
  loopcnt++;
  digitalWrite(OutputPin, (loopcnt / 10) % 2);
  if (pause_printing()) {
    return; // nichts ausgeben damit der Graph stehen bleibt
  }
  print_zeitraster();
  // Kanaele lesen und ausgeben
  for (int i = 0; i < 4; i++) {
    Serial.print(digitalRead(Pin[i]) + (i + 1) * 2);
    Serial.print(" ");
  }
  Serial.println("");
}
```

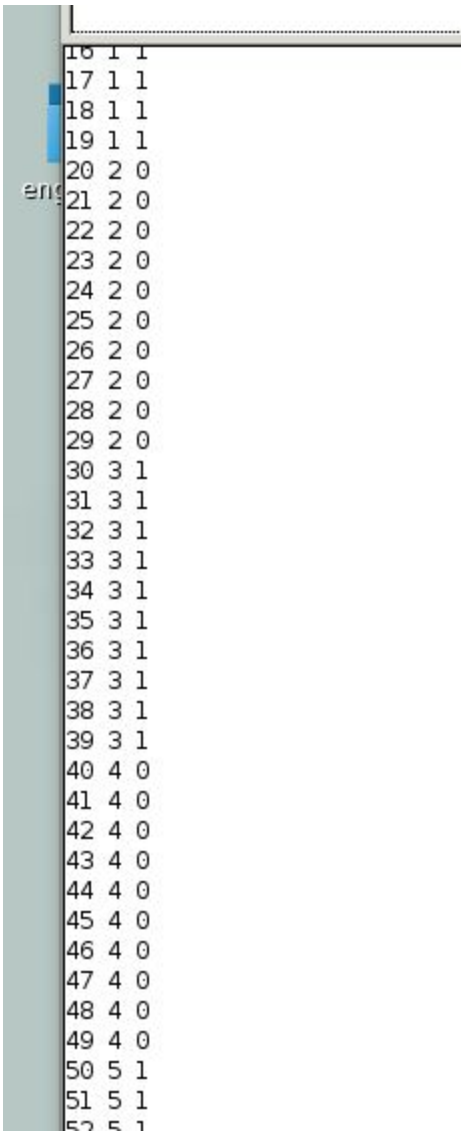
UND DOCH NOCH - RECHTECKAUSGABE

Beim Einfügen des Programms fiel der digitalWrite-Code auf.

```
void loop() {
  static int loopcnt = 0;
  loopcnt++;
  digitalWrite(OutputPin, (loopcnt / 10) % 2);
```

1. "`int loopcnt`" Es wird Speicherplatz für eine Ganzzahl, "`int`", Variable reserviert und diesem der Name "`loopcnt`" zugewiesen.
2. "`static`" hat zur Folge, dass nur einmal beim Programmstart der Wert auf "`0`" gesetzt wird. Ansonst werden Variablen die innerhalb einer Funktion deklariert werden bei jedem Aufruf der Funktion neu initialisert, hier "`= 0`". Damit würde "`loopcnt`" jedesmal am Anfang auf "`0`" gesetzt und einmal um eins erhöht. "`static`" ermöglicht das Zählen der Funktionsaufrufe.
3. "`loopcnt++;`" Der Wert von "`loopcnt`" wird bei jedem Durchlauf um eins erhöht.
4. "`(loopcnt / 10) % 2`" - "`%`" ist der Modulo-Operator. Dieser liefert den Rest der Division. "`x % 2`" ist 0 für gerade "`x`" und 1 für ungerade.
   "`(loopcnt / 10) % 2`" ist 0 für "`loopcnt`"-Werte zwischen 0 und 9, 20 und 29, 40 und 49 und 1 für Werte zwischen 10 und 19, und so weiter.
   Das bedeutet der Ausgang schaltet mit einem zwanzigstel der Frequenz des Aufrufs der Funktion "`loop`".
5. Es tut mir leid wenn ich das jetzt sehr kompliziert erklärt habe, probieren ist besser.

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  static int loopcnt = 0;
  loopcnt++;
  Serial.print(loopcnt);
  Serial.print(" ");
  int loopcnt10tel = loopcnt / 10;
  Serial.print(loopcnt10tel);
  Serial.print(" ");
  int loopcnt_is_odd = loopcnt10tel % 2;
  Serial.print(loopcnt_is_odd);
  Serial.println(" ");
  delay(1000);
}
```



RECHTECKAUSGABE OBJEKT ORIENTIERT

Der Code

```
void loop() {
  static int loopcnt = 0;
  loopcnt++;
  digitalWrite(OutputPin, (loopcnt / 10) % 2);
```

definiert so etwas in der Art, das man im Programmiererjargon Objekt nennt.

Eine Funktion mit ihrem privaten Datensatz.

Anders als bei einem Objekt, kann auf diese Art aber nur ein Pin und eine Frequenz

ausgegeben werden. In C++ können wir eine Klasse deklarieren:

```
class SquareWave {
    int Pin;
    int Cnt;
  public:
    SquareWave(int p) {
      Pin = p;
      Cnt = 0;
      pinMode(Pin, OUTPUT);
    }
    void NextStep() {
      Cnt++;
      digitalWrite(Pin, (Cnt / 10) % 2);
    }
};

SquareWave sqw(LED_BUILTIN);
void setup() {
}
void loop() {
  sqw.NextStep();
  delay(200);
}
```

- Die Klasse "`SquareWave`" hat zwei private Variablen die "`Pin`"nummer und den Zählerstand "`Cnt`". Diese sind `privat`, also nur innerhalb des Objektes sicht- und verwendbar.
- Alles nach der Zeile "`public:`" ist auch von extern sichtbar, verwendbar.
- Die Funktion "`SquareWave`" wird als Constructor bezeichnet, sie wird hier "`SquareWave sqw(LED_BUILTIN);`" aufgerufen. Danach haben wir ein Objekt "`sqw`" mit den privaten Variablen "`Cnt`" auf 0 und "`Pin`" auf den übergebenen Wert "p" gesetzt.
- Die Methode "`NextStep`" wird hier "`sqw.NextStep();`" aufgerufen. Es wird "`Cnt`" um eins erhöht und danach der Pin gesetzt.

Was bringt das ?

Man kann Rechtecksignale auf mehreren Pins gleichzeitig ausgeben.

```
SquareWave sqw(LED_BUILTIN);
SquareWave sqx(12);
SquareWave sqy(11);

void setup() {
}
void loop() {
  sqw.NextStep();
  sqx.NextStep();
  if (digitalRead(LED_BUILTIN)) {
    sqy.NextStep();
  }
  delay(200);
}
```

AUFGABEN

1. Verschiedene Rechtecksignale ausgeben ... wie ?
2. Die Rechteckausgabe im logicanalyser objektorientiert ausführen.

REFERENCES

REVISION HISTORY

- 0.1 23.11.2020 - pre processor.