Letzte Woche haben wir uns zu der Umwandlung von AST zu RTLIL die Umsetzung von Blocking- und Non-Blocking Assignments in Prozessen (always-Blöcken) angeschaut.
Aufgabe war es, die Vorgehensweise von Yosys mithilfe des Quellcodes weiter zu verstehen.

Wdh.:
Bei Blocking-Assignments wird die Zuweisung direkt durchgeführt. Das bedeutet gleichzeitig, dass es mehrere Versionen eines Signals geben muss, da Blocking Assignments mehrfach nacheinander durchgeführt werden können.

Bei Non-Blocking Assignments wird bei mehrfachen Zuweisungen immer nur die neuste Zuweisung berücksichtigt. Alte Zuweisungen verfallen.

Yosys unterscheidet Assignments mit zwei verschiedenen AstNodes: AST_ASSIGN_LE für Non-Blocking Assignmments und AST_ASSIGN_EQ für Blocking Assignments.

Die Funktion void processAst(AstNode *ast) in der genRTLIL.cc Datei identifiziert die Nodes anhand des Typs und führt für die beiden Zuweisungsarten die im Yosys-Manual beschriebenen Schritte aus.
Auffällig ist dabei, dass die Schritte für Blocking- und Non-Blocking Assignments weitestgehend gleich sind. Es gibt nur einen Unterschied: Temporäre Signale werden zwar bei beiden Zuweisungsarten gelöscht, jedoch bei Blocking Assignments vorher noch einmal zwischengespeichert.

Da die Behandlung der Zuweisungen nur diesen Unterschied hat, wird sie in dem Case für Non-Blocking-Assignments AST_ASSIGN_LE beschrieben. Der Case für Blocking Assignments ist leer. Weil kein break; in dem Case steht, geht das Programm auch bei Blocking-Assignments in den Case AST_ASSIGN_LE:

```
case AST_ASSIGN_EQ:
    case AST_ASSIGN_LE:
      {
```

Die Assign AstNodes bekommen vom Parser als Kinder die linke und rechte Seite einer Zuweisung (children[0], children[1]). Aus der linken Seite und rechten Seite der Zuweisung wird dann RTLIL Code generiert (Die AstNode Struktur hat eine Funktion genRTLIL() die sich rekursiv aufruft).
Der RTLIL Generator für die rechte Seite bekommt außerdem noch als Parameter die Wortlänge der linken Seite, weil Zuweisungen nur mit gleicher Länge möglich sind.
Hier werden außerdem die Werte rechts in einer stackmap (Mapmodifikation zum speichern / wiederherstellen des aktuellen Zustands) gespeichert. Das Speichern ist relevant für Blocking-Assignments.

Anschließend werden alle Zuweisungen für die gleiche linke Seite gelöscht.
Aus den aktuellen lvalue und rvalue Werten wird ein Paar gebildet und auf einen Stapel *actions* gelegt.

Für Blocking-Assignments gibt es noch die folgende Erweiterung innerhalb des gleichen Case:

```
if (ast->type == AST_ASSIGN_EQ) {
          for (int i = 0; i < GetSize(unmapped_lvalue); i++)
            subst_rvalue_map.set(unmapped_lvalue[i], rvalue[i]);
}
```

subst steht hier wahrscheinlich für substitute und bezeichnet die map mit temporären rvalues für Blocking-Assignments. Mit set wird ein Zustand innerhalb der map erzeugt. So werden also

die rvalue Werte vor dem Löschen zwischengespeichert.

Die Cases und if-Statement Behandlung ist noch komplexer. Das wäre unteranderem der nächste Schritt. Ich würde aber trotzdem noch weiter versuchen, die Blocking- und Non-Blocking Assignments im Detail zu verstehen. Hier werden Strukturen genutzt (Hashlib, maps..) die ich noch nicht kenne und die anscheinend nicht so einfach zu verstehen sind. Die Kombination von Blocking- und Non-Blocking Assignments macht es außerdem kompliziert nachzuvollziehen, welche Code-Zeile für welche Struktur gebraucht wird.