Technische Universität Berlin

Fachgebiet Software and Embedded Systems Engineering
– Prof. Dr. Sabine Glesner
www.sese.tu-berlin.de Sekretariat TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin

Software Engineering eingebetteter Systeme WS 2021/22

Prof. Dr. Sabine Glesner **Verena Klös, Paul Kogel, Julian Klein**

2. Hausaufgabe (15 Punkte) **Bearbeitungsbeginn: 17.01. Abgabe: 17.02.**

*Dies ist die zweite von zwei bewerteten Hausaufgaben. Sie muss in Gruppen von jeweils **2 Studierenden** bearbeitet werden. Eine **alleinige** Bearbeitung ist nicht möglich. **Ihr müsst die Hausaufgabe in der Gruppe bearbeiten, die ihr auf ISIS gewählt habt.***

In dieser Aufgabe sollt ihr das Pflanzenversorgungssystem aus der 1. Hausaufgabe mit SystemC modellieren. Eure Implementierung soll die gesamte Spezifikation der 1. Hausaufgabe umsetzen. Wir haben die Definition der Signale für die Verwendung in SystemC angepasst. Eine genaue Beschreibung der Signale findet ihr im Abschnitt *Testbench*.

Wir geben für diese Aufgabe ein CMAKE-Projekt vor. Das findet ihr auf ISIS zum Download. Ihr müsst alle Codedateien (*.cpp* und *.h*) eurer Lösung im Ordner *src* in unserem Projekt speichern.

Bei folgendem Verhalten werten wir eure Abgabe mit **0 Punkten**:

- Ihr verändert unsere Testbench oder *cmake* Konfiguration (Datei `CMakeLists.txt`).
- Ihr verwendet Bibliotheken, mit Ausnahme von SystemC und der C++ Standardbibliothek.
- Eure Abgabe lässt sich nicht ohne Fehler kompilieren.

Wir **bewerten** eure Lösung automatisiert und gestaffelt:

- Grundlegende Funktionalität vorhanden: 8 Punkte
- Grundlegende Funktionalität vorhanden und Randfälle abgedeckt: bis zu 7 Punkte

Unsere Vorgabe enthält ein *Testbench* Modul mit Tests, mit denen ihr prüfen könnt, ob euer Programm unsere Grundanforderungen erfüllt. Sind die grundlegenden Tests erfolgreich, erhaltet ihr 8 Punkte. **Schlägt mindestens ein Testfall fehl, werten wir eure Abgabe mit 0 Punkten.**

Erfüllt eure Lösung unsere Grundanforderungen, könnt ihr **bis zu 7 weitere Punkte** erhalten. Dazu muss euer Modell die Spezifikation auch in Randfällen präzise umsetzen. Dies testen wir mit weiteren Tests. Diese Tests veröffentlichen wir nicht. Ihr erhaltet Punkte für jeden abgedeckten Randfall.

Allgemeine Hinweise

Tools

Für diese Aufgabe benötigt ihr SystemC und *cmake*.

- Auf ISIS findet ihr eine virtuelle Maschine zum Download, die alle benötigten Tools enthält. Alternativ könnt ihr euch auch eine eigene Entwicklungsumgebung einrichten. Eine Anleitung zur Installation von SystemC unter Ubuntu und Windows findet ihr ebenfalls auf ISIS.
- Eine Anleitung zum Kompilieren des Projektes und Ausführen der Tests findet ihr im Anhang dieses Aufgabenblattes.

Abgabe

Verpackt zur Abgabe das CMAKE-Projekt als `.zip`-Datei und gebt diese auf ISIS ab. Abgaben müssen bis spätestens 17.02.2021 um 23:55 erfolgen. Einreichungen per Mail werden **nicht** berücksichtigt!

Täuschungsversuche/Plagiate

Jeder Versuch, die Leistung einer Person, die nicht zu eurer Gruppe gehört, als eure eigene Leistung auszugeben, ist ein Täuschungsversuch. Dazu zählt beispielweise:

- Mit einer anderen Gruppe zusammenarbeiten
- Code/Modelle einer anderen Gruppe teilweise oder vollständig übernehmen

Plagiate sind **unfair** allen ehrlichen Studierenden gegenüber. Wir setzen moderne Software ein, die solche Plagiate in euren Abgaben findet. Habt ihr plagiiert, fallt ihr **sofort mit einer 5.0** durch das Modul. Die Corona-Freiversuchsregelung gilt bei Täuschungsversuchen nicht.

Hilfe erhalten

Eine Spezifikation in natürlicher Sprache ist nicht immer eindeutig. Habt ihr Fragen zum Verständnis der Hausaufgabe oder auch zur allgemeinen Organisation, könnt ihr uns auf vielfältige Weise erreichen. **Der beste Ort für eure Fragen ist unser ISIS Forum zur Hausaufgabe. Stellt eure Fragen am besten zuerst dort, da so auch alle anderen Gruppen von den Antworten profitieren!** Außerdem heißen wir euch in unseren wöchentlichen Sprechstunden willkommen. Fragen per Mail beantworten wir nicht.

Testbench

Wir geben euch den Einstiegspunkt für die Simulation (Datei `main.cpp`) und eine Testbench (Dateien `TestBench.h`, `TestBench.cpp`, `TestHelper.h` und `TestHelper.cpp`) vor. Die Testbench prüft, ob eure Implementierung die Spezifikation korrekt umsetzt:

- Die Testbench erzeugt verschiedene Signalverläufe, die mögliches Verhalten der Umgebung simulieren.
- Eure Implementierung soll auf diese Signale geeignet reagieren. Dazu sollt ihr Signale der Testbench empfangen und geeignete Steuerungsbefehle an diese senden.
- Anschließend prüft unsere Testbench anhand der empfangenen Befehle, ob ihr die Spezifikation korrekt implementiert.

Unsere Testbench **erzeugt** folgende Ausgabesignale:

Portname	Art	Datentyp	Beschreibung	Initialwert
`regen`	Signal	`bool`	`true`, solange es regnet, sonst `false`.	`false`
`temperatur`	Signal	`int`	Aktuelle Außentemperatur. Kann sofort ausgelesen werden.	`20`
`wetter`	Signal	`TestBench::WetterTyp`	Aktuelles Wetter	`sonnig`
`salatfeld_blattlaeuse`	FiFo	`bool`	Das Salatfeld wurde von Blattläusen befallen.	
`salatfeld_schnecken`	FiFo	`bool`	Das Salatfeld wurde von Schnecken befallen.	
`kuerbisfeld_blattlaeuse`	FiFo	`bool`	Das Kürbisfeld wurde von Blattläusen befallen.	
`wassertank_auffuellen`	FiFo	`int`	Menge in *ml*, um die der Wassertank aufgefüllt wird. Kann beliebiger positiver Wert sein.	

Die Testbench **erwartet** folgende Eingabesignale:

Portname	Art	Datentyp	Beschreibung	Initialwert
`salatfeld_bewaessern`	FiFo	`bool`	Das Salatfeld soll bewässert werden.	
`salatfeld_duengen`	FiFo	`bool`	Das Salatfeld soll gedüngt werden.	
`salatfeld_pflanzenschutz`	FiFo	`bool`	Auf dem Salatfeld wird Pflanzenschutz versprüht.	
`kuerbisfeld_bewaessern`	FiFo	`bool`	Das Kürbisfeld soll bewässert werden.	
`kuerbisfeld_duengen`	FiFo	`bool`	Das Kürbisfeld soll gedüngt werden.	
`kuerbisfeld_pflanzenschutz`	FiFo	`bool`	Auf dem Kürbisfeld wird Pflanzenschutz versprüht.	
`gewaechshaus_bewaessern`	FiFo	`bool`	Im Gewächshaus soll bewässert werden.	
`gewaechshaus_duengen`	FiFo	`bool`	Im Gewächshaus soll gedüngt werden.	
`gewaechshaus_heizung`	Signal	`int`	Aktuelle Stufe der Heizung im Gewächshaus.	`1`
`gewaechshaus_lampe`	Signal	`int`	Aktuelle Stufe der Lampe im Gewächshaus.	`0`
`wassertank_warnlampe`	Signal	`bool`	`true`, wenn die Warnlampe am Tank leuchten soll, sonst `false`.	`true`

In der Vorgabe haben wir bereits jeden Ein- und Ausgabeport der Testbench mit einem eigenen Kanal verbunden. Diese Kanäle und Verbindungen sind in `main.cpp` definiert. **Um mit der Testbench zu kommunizieren, müsst ihr eure Module ebenfalls mit diesen Kanälen verbinden**:

- Schließt die Ports eurer Module an die passenden Kanäle an. Passt dazu `main.cpp` an. **Jeder Port muss mit einem Kanal verbunden sein!** Ansonsten kann SystemC euer Programm nicht kompilieren.
- Jeder Kanal kann mit maximal einem Ausgabeport (= schreibender Prozess) verbunden werden. FiFo-Kanäle können zusätzlich nur mit einem Eingabeport (= lesender Prozess) verbunden werden, bei Signal-Kanälen sind mehrere Eingabeports möglich. Benötigt ihr **mehrere lesende oder schreibende Prozesse**, empfehlen wir einen *Koordinator*:
  - Verbindet einen Prozess mit dem Kanal. Das ist der Koordinator.
  - Der Koordinator schreibt Werte in den Kanal, die er von anderen Prozessen empfängt bzw. leitet gelesene Werte an diese Prozesse weiter.
  - Der Koordinator und die anderen Prozesse kommunizieren über *lokale Events*. Wie ihr lokale Events verwenden könnt, ist im Anhang beschrieben.
- Signale müssen von dem Modul initialisiert werden, das sie schreibt. Ihr müsst also selbst alle **Eingabesignale** vom Typ *Signal* mit dem geforderten Initialwert **initialisieren**. Sonst können Tests fehlschlagen.

Wichtige Hinweise zu FiFo-Kanälen *(Queues)*:

- Unsere Queues sind zu Beginn leer, besitzen also keinen Initialwert.
- Wir verwenden Queues mit Datentyp `bool` als Event-Buffer: es werden Elemente eingefügt, wenn das zugehörige Ereignis (Schneckenbefall, Blattläuse etc.) eintritt. Der Wert der eingefügten Elemente spielt keine Rolle, wird ein Element eingefügt, wird immer angenommen, dass das entsprechende Ereignis eingetreten ist.
- Unsere Queues speichern maximal 1 Element. Befindet sich bereits ein Element in der Queue, kann dies nicht überschrieben werden. Haltet daher unbedingt folgende Punkte ein:
  - Entnehmt immer alle neuen Werte. Ansonsten schlagen Tests fehl.
  - Verwendet ausschließlich `write_nb`, um Werte in FiFo-Kanäle zu schreiben. Ansonsten schlagen Tests fehl.

Anhang

Kompilieren und Testen mit CMAKE

Wir nutzen *cmake* zum Kompilieren und Testen des Codes. Verwendet folgende Befehle, um eure Implementierung zu kompilieren und unsere mitgelieferten grundlegenden Tests auszuführen:

a) Öffnet ein Terminal. Wechselt in das Verzeichnis, das `CMakeLists.txt` enthält.

b) **Konfiguriert** das Projekt: Befehl `cmake .`

c) **Kompiliert** das Projekt: Befehl `make`

d) **Testet** eure Implementierung: Befehl `ctest`

Bei wiederholtem Kompilieren und Testen müsst ihr den 2. Schritt nicht erneut ausführen.

Standardmäßig unterdrückt `ctest` Konsolenausgaben.

- Verwendet `ctest -V`, um alle Ausgaben anzuzeigen.
- Verwendet `ctest --output-on-failure`, um nur die Ausgaben der fehlgeschlagenen Tests zu sehen.

Ihr könnte auch Tests einzeln ausführen.

- Verwendet dafür `./ha2` und dann den Namen des Tests.
- Also zum Beispiel `./ha2 testBewaesserungOhneRegen` um den ersten Test auszuführen.

SystemC Hilfe

Nützliche Befehle, die nicht in der Übung vorkamen:

Lokale Events

Ihr könnt lokale Events zur modulinternen Prozesssynchronisation verwenden.

- `sc_event myEvent`: deklariert das lokale Event `myEvent`
- `myEvent.notify()`: löst `myEvent` aus
- `wait(myEvent)`: wartet auf `myEvent`

Warten auf mehrere Events und Timeouts

- `wait (A | B | ...)`: warte, bis A oder B eintritt (oder weitere). Hinterher müsst ihr ggf. auf geeignete Art und Weise prüfen, welches Event den Prozess aktiviert hat.
- `wait (10, SC_SECONDS, A)`: warte maximal 10 Sekunden auf A. Hinterher müsst ihr ggf. prüfen, wie viel Zeit während des Wartens vergangen ist.

Zeit

- `sc_time_stamp()`: gibt die aktuelle Simulationszeit als `sc_time` Objekt an. Ihr könnt `sc_time` Objekte addieren und subtrahieren.
- `to_seconds()`: liefert die Sekunden eines `sc_time` Objektes als Double.
  Beispiel: `double now = sc_time_stamp().to_seconds()`