ingo.korb [☺] tu-dortmund.de
thomas.hess [☺] tu-dortmund.de
niklas.ueter [☺] tu-dortmund.de

Übung zur Vorlesung
Eingebettete Systeme
Wintersemester 15/16

Übungsblatt 12

(10 Punkte)

Präsenzaufgaben zur Woche ab Monday, February 1, 2016

Für die Bearbeitung dieses Blatts wird die virtuelle Maschine **RTEMS-de** verwendet.

Da unsere Lizenz für eines der verwendeten Tools auf bestimmte Rechner beschränkt ist, müssen Sie sich aus dieser VM heraus per SSH zum Zielrechner verbinden. Es sollte dabei keine erneute Passworteingabe notwendig sein. Zur Verbindung mit dem Zielrechner verwenden Sie bitte das Kommando **ssh -X ls12pc5.cs.tu-dortmund.de** in einem Terminalfenster.

12.1 Schritt 3: Scratchpad-Allokation und Function Outlining (5 Punkte)

Verwenden Sie den Befehl `cd ~wcet/step3` um in das Verzeichnis für diese Teilaufgabe zu wechseln. In der Datei `test.c` wurden einige Vorbereitungen getroffen, um die inneren Schleifen der Funktionen `Initialize` und `Sum` per Function Outlining in getrennte Funktionen abzuspalten, damit selektiv nur diese inneren Funktionen in das Scratchpad Memory (SPM) verschoben werden können.

Nehmen Sie das Function Outlining für die inneren Schleifen dieser beiden Funktionen vor.

Compilieren Sie danach das Beispielprogramm mit dem Befehl **tricore-gcc -o test.elf -g -T ../tc1796.lds test.c** und laden Sie es wie beim vorherigen Übungsblatt in den `a3tricore`-Analyzer. Wenn Sie nun eine aiT-Analyse starten, werden bei der Analyse einige Fehler gemeldet.

Korrigieren Sie die Fehler in der Datei `a.ais`, damit die Analyse erfolgreich durchgeführt werden kann.

Wie hat sich die WCET im Vergleich zur vorherigen Fassung (komplette Funktionen im SPM) verändert?

Tip: Dort hatte der Analyzer eine WCET von 85117 Zyklen berechnet.

12.2 Schritt 4: Integer-Lineare Programmierung (5 Punkte)

aiT analysiert die WCET eines Programmes, indem es jeden Basisblock einzeln analysiert, mit den Ergebnissen ein ILP (Integer-Lineares Programm) aufstellt und dieses löst.

Im Verzeichnis `step4` befindet sich eine Beispielformulierung eines ILPs in der Datei `example.lp`. Schauen Sie sich den Inhalt dieser Datei mit einem beliebigen Texteditor an und lassen Sie es mittels `lp_solve example.lp` von einem ILP-Solver lösen.

Als weiteres Beispiel befindet sich im Verzeichnis die Datei `example2.lp`, die den Kontrollflussgraphen eines sehr einfachen Programms modelliert. Auch diese Datei können Sie mit `lp_solve example2.lp` lösen lassen.

Erläutern Sie das Ergebnis.

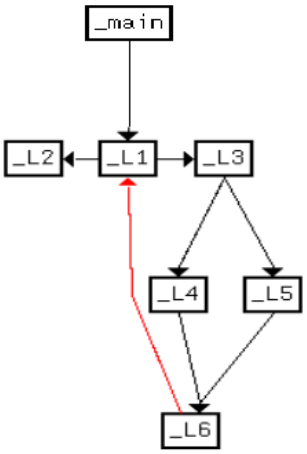
Wie kann man im ILP modellieren, dass ein Basisblock eine Laufzeit von mehr als einem Zyklus hat?

Gegeben sei nun folgendes kleines Programm mit Schleife:

```
int main()
{
  int i, j = 0;

  _Pragma( "loopbound min
                100 max 100" );
  for ( i = 0; i < 100; i++ ) {
    if ( i < 50 )
      j += i;
    else
      j += ( i * 13 ) % 42;
  }

  return j;
}
```



Block	Zyklen
main	21
L1	27
L2	20
L3	2
L4	2
L5	20
L6	13

Abbildung 1: Ein Beispielprogramm mit Schleife

Die grundsätzliche Struktur dieses Programms ist bereits in der Datei `ipet.lp` modelliert.

Ergänzen Sie in der Datei die Anzahl der Zyklen für jeden Basisblock.

Wenn Sie nun versuchen, das ILP per `lp_solve ipet.lp` lösen zu lassen, wird der Solver mit der Fehlermeldung *This problem is unbounded* abbrechen.

Warum kann der ILP-Solver keine Lösung finden?

Ergänzen Sie den fehlenden Constraint für Kante h.

Lassen Sie die korrigierte Datei nochmals von lp_solve lösen – Sie sollten nun ein Ergebnis erhalten.
Warum berechnet der Solver eine Lösung, in der L4 nie ausgeführt wird?