Hochschule Osnabrück
Fakultät für Ingenieurwissenschaften und Informatik
Labor für Digital- und Mikroprozessortechnik
Prof. Dr.-Ing. B. Lang, Dipl.-Inf. (FH) R. Höckmann

Praktikum Rechnerorganisation

Versuch 5: Erweiterte Stoppuhr mit serieller Schnittstelle

Lernziele

- Einbinden einer UART-Komponente in das Beispielrechner-System.
- Verwendung mehrerer Interrupts
- Kommunikation zwischen Hauptprogramm und Interrupt-Handler

Verwendete Werkzeuge

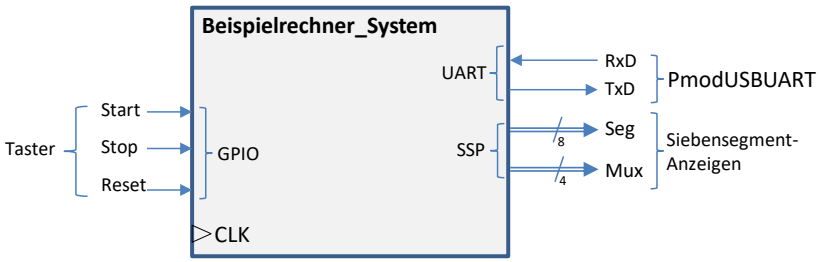
- BASYS3-Board
- PModUSBUART-Adapter
- ModelSim (Intel FPGA Starter Edition)
- Vivado (Version 2019.2)
- Eclipse IDE for C/C++ Developers (Version 2020-12)
- GNU-Toolchain für MIPS
- HTerm

Vorbereitung (Vor dem Praktikumstermin)

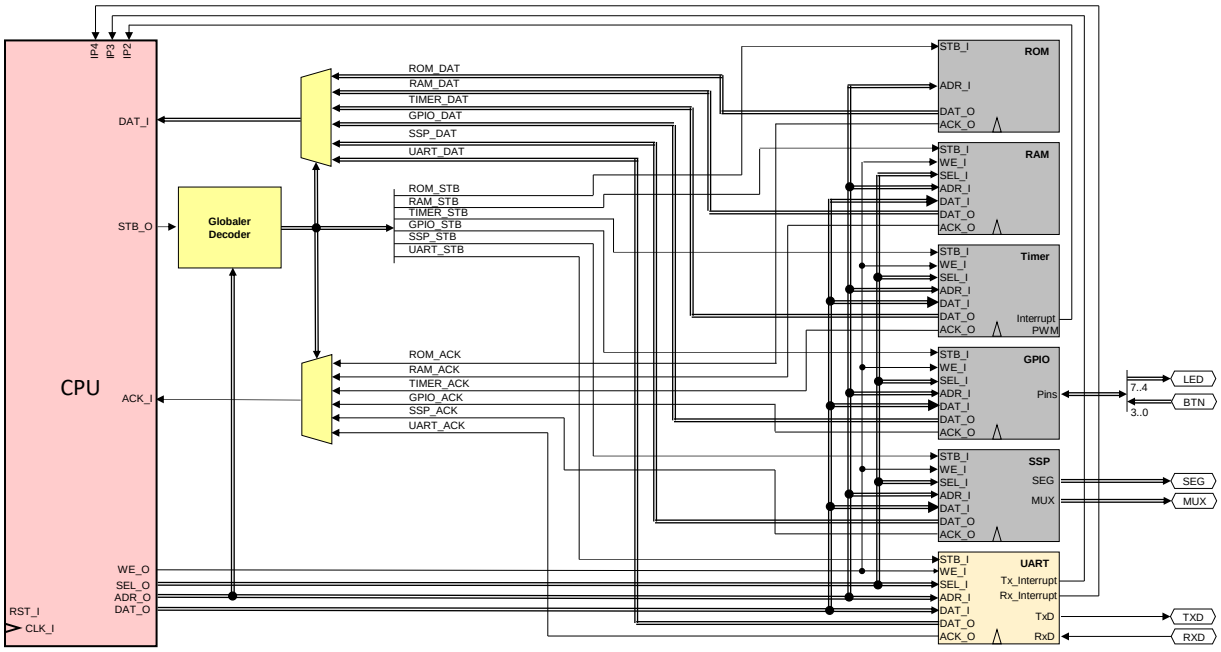
- Arbeiten Sie die Abschnitte des Skripts über den UART die Interrupt-Verarbeitung gründlich durch.
- Führen Sie die Punkte zur Vorbereitung der einzelnen Aufgaben durch.

Systembeschreibung

In diesem Praktikumsversuch soll die im vorherigen Versuch mit dem Beispielrechner realisierte Stoppuhr so erweitert werden, dass sie über eine asynchrone serielle Schnittstelle mit einem PC verbunden und über diesen auch bedient werden kann. Da die meisten aktuellen PCs heute ohne COM-Port (Communication Port) ausgeliefert werden, wird ein Adapter verwendet, der die asynchrone serielle Schnittstelle des Beispielrechner-Systems per USB (Universal Serial Bus) mit dem PC verbindet. Auf dem PC richtet ein zugehöriger Treiber einen virtuellen COM-Port ein. Ein Terminalprogramm kann sich mit diesem virtuellen COM-Port verbinden. Auch im Terminalfenster soll dann die aktuelle Zeit der Stoppuhr ausgegeben werden. Weiterhin soll per Tastatureingabe der Zeichen 's', 'x' und 'r' im Terminalfenster die Stoppuhr gestartet, gestoppt und rückgesetzt werden können.



Das Beispielrechner-System vom letzten Versuch wird dazu um eine UART-Komponente erweitert. Die folgende Abbildung zeigt das erweiterte System:

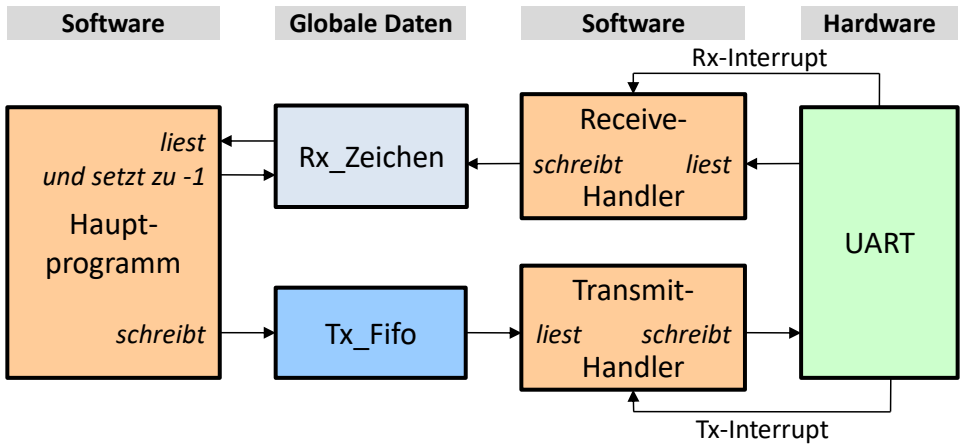


Software

In **Aufgabe 1** wird eine Testsoftware erstellt, welche Interrupt-gesteuert ein empfangenes Zeichen vom UART liest, die hexadezimale Darstellung seines Werts berechnet und diese Zeichenkette über ein Software-FIFO (First In, First Out) wieder auf den UART ausgibt. Dabei werden Unterprogramme erstellt, welche den UART auf einfache Weise unterstützen. Ein C-ähnlicher Code erläutert die Funktion dieser Software.

In **Aufgabe 2** wird die in Versuch 4 erstellte Stoppuhr-Software so modifiziert, dass die Stoppuhr die erweiterte Funktionalität erhält. Dabei werden bereits erstellte Unterprogramme wiederverwendet. Ein C-ähnlicher Code erläutert die Funktion der erweiterten Stoppuhr.

In beiden Aufgaben erfolgt die Kommunikation des Hauptprogramms mit dem UART in Empfangsrichtung über eine globale Variable *Rx_Zeichen* und in Senderichtung über ein Software-FIFO *Tx_Fifo*. Diese Kommunikation wird durch die Interrupt-Handler unterstützt (siehe folgende Abbildung).

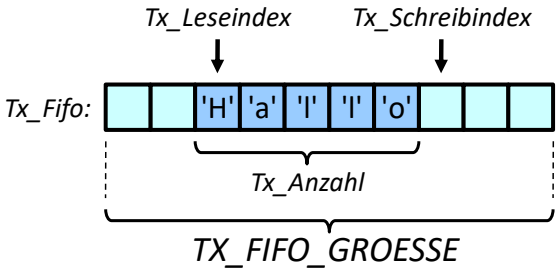


Hauptprogramm: Liest die empfangenen Zeichen mit der Funktion *Rx_Zeichen_holen* aus der globalen Variablen *Rx_Zeichen*, welche dabei zu -1 gesetzt wird. Ist der gelesene Wert ungleich -1, liegt ein empfangenes Zeichen vor und kann verarbeitet werden.

Auszugebende Zeichen werden mit der Funktion *Tx_Fifo_schreiben* in das Software-FIFO geschrieben, wobei auch der Sende-Interrupt Tx_Interrupt freigegeben wird.

Beim Zugriff auf das FIFO und auf die globale Variable *Rx_Zeichen* gibt es kritische Code-Abschnitte, in denen bestimmte Interrupts nicht erlaubt sind. Diese Abschnitte werden dadurch geschützt, dass vorher der jeweils verbotene Interrupt gesperrt und im Anschluss wieder freigegeben wird.

Transmit-Handler: Bei aktivem Tx-Interrupt des UART wird versucht, ein Zeichen aus dem Software-FIFO zu lesen. Bei Erfolg wird das gelesene Zeichen in den UART geschrieben. Konnte kein Zeichen gelesen werden, wird der Sende-Interrupt gesperrt.
Receive-Handler: Bei aktivem Empfangs-Interrupt Rx_Interrupt des UART wird das empfangene Zeichen aus dem UART ausgelesen und in die globale Variable Rx_Zeichen geschrieben.
Timer-Handler: Auch im erweiterten Stoppuhr-Programm wird natürlich weiterhin der Timer-Interrupt unterstützt.
Software-Fifo: Das FIFO wird durch einen Schreibzähler *Tx_Schreibindex*, einen Lesezähler *Tx_Leseindex*, einen Zähler *Tx_Anzahl* für die Anzahl im FIFO aktuell gespeicherten Zeichen beschrieben. Die im FIFO gespeicherten Daten liegen in einem Feld *Tx_Fifo* der konstanten Größe *TX_FIFO_GROESSE* beschrieben. Die Variable *Tx_Leseindex* enthält die Position im Feld, an der das nächste Zeichen gelesen wird. Ebenso zeigt *TX_Schreibindex* auf die Position im Feld *Tx_Fifo*, an der das nächste Zeichen geschrieben wird. Nach dem Schreiben bzw. Lesen wird der zugehörige Zeiger erhöht. Zeigt er danach hinter das FIFO, wird er zurück auf den Feldanfang, d.h. auf den Wert 0, gesetzt. Ob das FIFO geschrieben oder gelesen werden darf, hängt von der Anzahl der eingeschriebenen Zeichen ab. Bei *Tx_Anzahl* = 0 darf nicht gelesen und bei *Tx_Anzahl* = *TX_FIFO_GROESSE* nicht geschrieben werden. Die folgende Abbildung verdeutlicht nochmals diese Zusammenhänge.



Aufgabe 1: Erstellung und Test des Systems mit UART

Lernziele

- Aufbau eines Systems mit externer Kommunikation
- Mechanismen zur Verwendung von Interrupts von mehreren Peripheriekomponenten
- Zwischenpufferung von Daten bei der Interrupt-Bearbeitung

Aufgabenstellung

Das Beispielrechnersystem soll wie oben beschrieben erweitert werden. Dazu muss der in Versuch 2 in VHDL erstellte UART-Sender dem UART hinzugefügt, dann der fertige UART in das Beispielrechnersystem eingebaut und schließlich Adressdecoder, Lesedatenmultiplexer und ACK-Multiplexer angepasst werden.

Das System soll anschließend mit einer Software Teste_UART.S getestet werden. Diese liest ein Zeichen vom UART, ermittelt seine hexadezimale Darstellung und gibt diese über den UART wieder aus. Schließlich werden noch ein Zeilenumbruch (‚\n') angehängt. Beispielsweise wird für das vom UART gelesene Zeichen `'A'`, dem die ASCII-Kodierung 0x41 zugeordnet ist, die Zeichenkette `"0x41\n"` über den UART ausgegeben.

Die Testsoftware Teste_UART.S wird durch die folgende, an die Sprache C angelehnte Softwarebeschreibung erläutert:

```
// ------------------------------------------------------------------------
// Peripherie-Definitionen
// ------------------------------------------------------------------------
#define UART_Basis          0x8300
#define UART_TxData         0
#define UART_RxData         4
#define UART_Kontroll       8
#define UART_Status         12

#define UART_TxD_IrEn       0
#define UART_RxD_IrEn       1
#define UART_TxD_OK         0
#define UART_RxD_OK         1
#define UART_RxD_Err        2

// ------------------------------------------------------------------------
// Anwendungsspezifische Definitionen
// ------------------------------------------------------------------------
#define TX_FIFO_GROESSE     16
```

```
// ------------------------------------------------------------------------
// Unterprogramme (aus Versuch 3 übernommen)
// ------------------------------------------------------------------------
void setze_bit(int Adresswert, int Bitnummer);
void loesche_bit(int Adresswert, int Bitnummer);

// Globale Variablen
volatile int Rx_Zeichen;
volatile int Tx_Schreibindex;
volatile int Tx_Leseindex;
volatile int Tx_Anzahl;
volatile char Tx_Fifo[TX_FIFO_GROESSE];
const char Bin_to_ASCII[] = "0123456789ABCDEF";


// Interrupt-Funktion zum Senden
void Transmit_Handler() {
  if (Tx_Anzahl > 0) { // Daten im Fifo
    // Zeichen aus Fifo lesen und in UART schreiben
    *((int*)(UART_Basis + UART_TxData)) = Tx_Fifo[Tx_Leseindex];
    Tx_Leseindex++;                      // Leseindex erhoehen
    if (Tx_Leseindex == TX_FIFO_GROESSE) {
      Tx_Leseindex = 0;
    }
    Tx_Anzahl--;
  } else { // Keine Daten mehr im Fifo, Tx-Interrupt sperren
    loesche_bit(UART_Basis + UART_Kontroll, UART_TxD_IrEn);
  }
}

// Interrupt-Funktion zum Empfangen eines Zeichens
void Receive_Handler() {
  Rx_Zeichen = *((char*)(UART_Basis + UART_RxData));
}

// UART-Funktion zum Holen eines empfangenen Zeichens
int Rx_Zeichen_holen() {
  int Zeichen;
  loesche_bit(UART_Basis + UART_Kontroll, UART_RxD_IrEn);
  Zeichen = Rx_Zeichen;
  Rx_Zeichen = -1;
  setze_bit(UART_Basis + UART_Kontroll, UART_RxD_IrEn);
  return Zeichen;
}

// UART-Funktion zum Schreiben des Sende-Fifos
void Tx_Fifo_schreiben(char Wert) {
  while (Tx_Anzahl == TX_FIFO_GROESSE) {
    // Warten auf Platz im Fifo
  }
  loesche_bit(UART_Basis + UART_Kontroll, UART_TxD_IrEn);
  Tx_Fifo[Tx_Schreibindex] = Wert;       // Wert in Fifo schreiben
  Tx_Schreibindex++;                     // Leseindex erhoehen
  if (Tx_Schreibindex == TX_FIFO_GROESSE) {
     Tx_Schreibindex = 0;
  }
  Tx_Anzahl++;
  setze_bit(UART_Basis + UART_Kontroll, UART_TxD_IrEn);
}

// Hauptprogramm
int main() {
  Rx_Zeichen = -1;


  // Interrupt IP3 freigeben
  // Interrupt IP4 freigeben
  // Globalen Interrupt freigeben

  // UART initialisieren
  setze_bit(UART_Basis + UART_Kontroll, UART_RxD_IrEn);
```

```
    while(1) {

      // Zeichen lesen
      int Zeichen = Rx_Zeichen_holen();

      // Zeichen konvertieren und ausgeben
      if (Zeichen >= 0) {
        // Wert des Zeichens als hexadezimale Konstante ausgeben
        Tx_Fifo_schreiben('0');
        Tx_Fifo_schreiben('x');

        int Hi = (Zeichen >> 4) & 0xf;
        Tx_Fifo_schreiben(Bin_to_ASCII[Hi]);

        int Lo = Zeichen & 0xf;
        Tx_Fifo_schreiben(Bin_to_ASCII[Lo]);

        Tx_Fifo_schreiben('\n');
      }
    }
  }
```

Die Hardwarebeschreibung des Systems mit der im VHDL-Speicher installierten Testsoftware wird mit einer Testbench getestet, welche nacheinander vier Zeichen an den seriellen Eingang RXD des Systems sendet, so dass die Reaktion des Systems auf dem seriellen TXD-Ausgang beobachtet werden kann.

Vorbereitung (Vor dem Praktikumstermin)

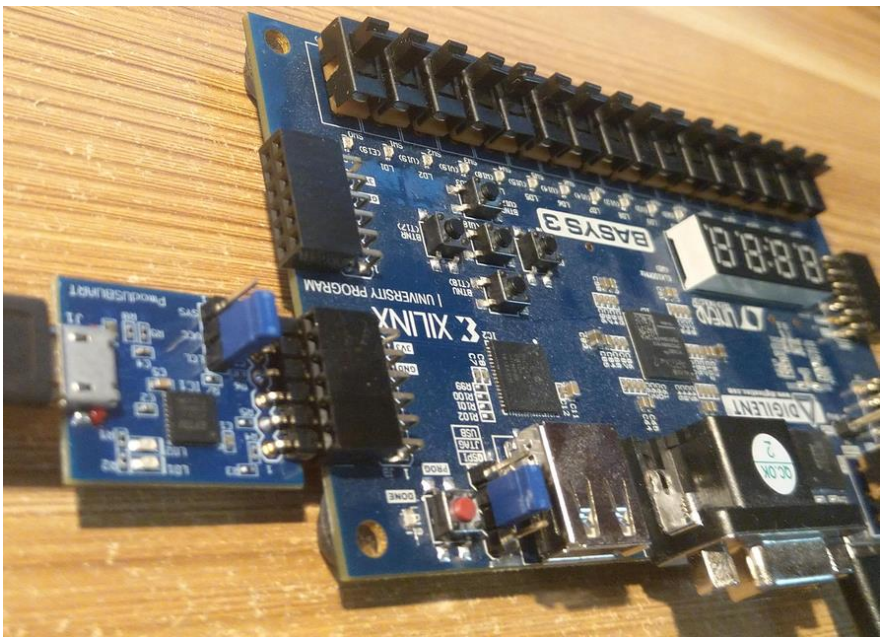
1. Ausgangspunkt ist das Beispielrechnersystem aus Versuch 4.
2. Ergänzen Sie die Verzeichnisstruktur durch die in diesem Versuch bereitgestellten Dateien. Beachten Sie, dass die neuen Dateien in die richtigen Verzeichnisse eingefügt werden.
3. Kopieren Sie die in Versuch 2 erstellte Datei UART_Sender.vhd in das Verzeichnis *Hardware\Peripherie*.
4. Passen Sie das Beispielrechner-System an die neue Systemarchitektur an. Instanziieren Sie dazu die Komponente *UART*. Der UART soll für eine Übertragungsrate von 115200 Baud konfiguriert werden (Hinweis: Die Frequenz des Systemtakts ist 60 MHz). Passen Sie die entity des Systems, den Adressdecoder sowie die Multiplexer für die Lesedaten und die ACKs der Komponenten an.
5. Machen Sie sich mit der oben gezeigten C-ähnlichen Beschreibung der Software vertraut.
6. Machen Sie sich mit der bereitgestellten Testbench Beispielrechner_System_V5_A1_testbench.vhd vertraut.

Durchführung

1. Starten Sie die Eclipse. Verwenden Sie als Workspace wieder das Verzeichnis *Software* im Arbeitsverzeichnis.
2. Importieren Sie das Projekt „Teste_UART“ in den Workspace. Dieses enthält im Verzeichnis *source* die Assembler-Datei Teste_UART.S.
3. Kopieren Sie die aus Versuch 3 bekannten Unterprogramme in das Programm (Stellen mit TODO markiert).
4. Ergänzen Sie im Programm an der markierten Stelle den fehlenden Assemblercode für den Interrupt-Handler Transmit_Handler.
5. Lassen Sie das das Programm übersetzen. Rufen Sie nach erfolgreicher Übersetzung die Batch-Datei CopyHexFile.bat auf.
6. Übersetzen und simulieren Sie das System mit der Testsoftware im VHDL-Simulator mit dem Script test_Beispielrechner_System_V5_A1.do. Falls beim Übersetzen Fehler auftreten, korrigieren Sie diese.
7. Inspizieren Sie das Wave-Fenster. Die Testbench sendet nacheinander mehrere Zeichen zum System, welches die zugehörigen ASCII-Kodierungen zurücksenden muss.
8. Falls die Testbench Fehler meldet, korrigieren Sie diese.
9. Starten Sie Vivado und öffnen Sie das Projekt Hardware\Synthese\Synthese.xpr.
10. Fügen Sie die neu hinzugefügten VHDL-Dateien der UART-Komponente dem Projekt hinzu (Add Source). Achten Sie darauf, die Datei nur zu referenzieren statt sie zu kopieren.
11. Ergänzen Sie in der Constraints-Datei Beispielrechner_System.xdc die Einträge für die neu hinzugekommenen Signale der seriellen Schnittstelle:

```
set_property -dict {PACKAGE_PIN A17 IOSTANDARD LVCMOS33} [get_ports {TXD}];
set_property -dict {PACKAGE_PIN C15 IOSTANDARD LVCMOS33} [get_ports {RXD}];
```

12. Lassen Sie Vivado eine Bitstream-Datei erzeugen (Generate Bitstream).
13. **Schalten Sie (falls eingeschaltet) das BASYS3-Board aus.** Stecken Sie das PmodUSBUART-Modul in die Erweiterungsbuchse JB des Basys3-Boards (untere Reihe, siehe Foto). Verbinden Sie das Basys3-Board und das PmodUSBUART-Modul mit jeweils einem USB-Kabel mit dem PC. Schalten Sie das Basys3-Board ein.
14. Übertragen Sie die Bitstream-Datei auf das FPGA
15. Starten Sie den BSR2-GDB-Server. Eventuell müssen Sie in der .ini-Datei noch den COM-Port anpassen.
16. Erzeugen Sie für die Applikation eine Debug-Konfiguration und führen Sie mit dieser das Programm auf dem Beispielrechner aus.
17. Ermitteln Sie im Windows-Gerätemanager den verwendeten COM-Port des PModUSBUART-Moduls.
18. Starten Sie das Terminalprogramm HTerm auf dem PC. Verbinden Sie es mit dem virtuellen COM-Port des PmodUSBUART-Moduls und stellen Sie die Kommunikationsparameter ein (115200 Baud, keine Parität, 1 Stoppbit).
19. Senden Sie einige Zeichen an das System und überprüfen Sie, dass jeweils eine Zeichenkette mit der ASCII-Kodierung zurückgesandt wird.


EASYS 3
XILINX
UNIVERSITY PROGRAM
DIGILENT

Aufgabe 2: Erweitertes Stoppuhr-System erstellen

Lernziele

- Erweiterung einer bestehenden Software
- Verwendung einer bestehenden Hardware mit der erweiterten Software

Aufgabenstellung

Es soll eine Software vervollständigt werden, welche die Ausgabe der Stoppuhr nicht nur auf der Siebensegmentanzeige ausgibt. Zur Steuerung sollen nicht nur die Taster auf dem Board sondern auch vom PC gesendete Zeichen dienen. Mit dem Zeichen `'s'` wird die Uhr gestartet. Das Zeichen `'x'` stoppt die Uhr. Mit dem Zeichen `'r'` wird die Uhr auf den Startwert zurückgesetzt. Bei jeder Änderung wird die aktuelle Uhrzeit auch über den UART in folgendem Format ausgegeben: „m:ss:z\n“, also z.B. „0:00,0\n“. Das Zeichen `'\n'` steht dabei für den Zeilenumbruch mit dem ASCII-Wert 0xA.

Die Software Erweiterte_Stoppuhr.S wird durch die folgende, an die Sprache C angelehnte Softwarebeschreibung erläutert:

```
// ------------------------------------------------------------------------
// Peripherie-Definitionen
// ------------------------------------------------------------------------
#define Timer_Basis         0x8000
#define Timer_Periode       0x0
#define Timer_Schwelle      0x4
#define Timer_Zaehlerstand  0x8
#define Timer_Kontroll      0xC
#define Timer_Status        0x10

#define Timer_IrEn          0

#define GPIO_Basis          0x8100
#define GPIO_Eingabe        0x0
#define GPIO_Ausgabe        0x4
#define GPIO_Richtung       0x8

#define SSP_Basis           0x8200
#define SSP_Wert0           0x0
#define SSP_Wert1           0x4
#define SSP_Wert2           0x8

#define UART_Basis          0x8300
#define UART_TxData         0x0
#define UART_RxData         0x4
#define UART_Kontroll       0x8
#define UART_Status         0x10

#define UART_TxD_IrEn       0
#define UART_RxD_IrEn       1
#define UART_TxD_OK         0
#define UART_RxD_OK         1
#define UART_RxD_Err        2

// ------------------------------------------------------------------------
// Anwendungsspezifische Definitionen
// ------------------------------------------------------------------------
// Spezifikation der GPIO-Pins
#define START               0
#define STOP                1
#define RUECKSETZEN         2

#define TX_FIFO_GROESSE     16

// Konstanten fuer die Ausfuehrung in Hardware (Echtzeit)
#define ZAEHLER_PERIODE, 59999 # Echtzeit (Ein Interrupt pro ms)
#define MS_PRO_INTERRUPT, 1    # Echtzeit (1 ms pro Interrupt)

// Konstanten fuer die Ausfuehrung im VHDL-Simulator (Beschleunigung x100)
//#define ZAEHLER_PERIODE, 29999 # Beschleunigt (2 Interrupts pro ms)
//#define MS_PRO_INTERRUPT, 50   # Beschleunigt (50 ms pro Interrupt)
```

```
// ------------------------------------------------------------------------
//  Unterprogramme (aus Versuch 3 übernommen)
// ------------------------------------------------------------------------
int bit_von(int Wert, int Bitnummer);
int lese_bit(int Adresswert, int Bitnummer);
void setze_bit(int Adresswert, int Bitnummer);
void loesche_bit(int Adresswert, int Bitnummer);
void schreibe_bitfeld(int Adresswert, int Wert, int Maske, int Shift);

// ------------------------------------------------------------------------
//  Unterprogramme (aus Versuch 4 übernommen)
// ------------------------------------------------------------------------
void Timer_Handler();

// ------------------------------------------------------------------------
//  Unterprogramme (aus Aufgabe 1 übernommen)
// ------------------------------------------------------------------------
void Transmit_Handler();
void Receive_Handler();
int Rx_Zeichen_holen();
int Tx_Fifo_schreiben(int Wert);

// ------------------------------------------------------------------------
// Globale Variablen
// ------------------------------------------------------------------------
volatile int ms;
volatile int Rx_Zeichen;
volatile int Tx_Schreibindex;
volatile int TX_Leseindex;
volatile int Tx_Anzahl;
char Tx_Fifo[TX_FIFO_GROESSE];
const char Bin_to_ASCII[16] = "0123456789ABCDEF";

void Zeit_anzeigen(int Zehner, int Einer, int Zehntel) {

  // (aus Versuch 4 übernommen)
  *((int*)(SSP_Basis + SSP_Wert3)) = Minuten | 0x10; // Dezimalpunkt an;
  *((int*)(SSP_Basis + SSP_Wert2)) = Zehner;
  *((int*)(SSP_Basis + SSP_Wert1)) = Einer   | 0x10; // Dezimalpunkt an;
  *((int*)(SSP_Basis + SSP_Wert0)) = Zehntel;

  Tx_Fifo_schreiben(Minuten + '0');
  Tx_Fifo_schreiben(':');
  Tx_Fifo_schreiben(Zehner + '0');
  Tx_Fifo_schreiben(Einer + '0');
  Tx_Fifo_schreiben(',');
  Tx_Fifo_schreiben(Zehntel + '0');
  Tx_Fifo_schreiben(Zehntel + '\n');
}

void main() {
  int Minuten = 0;
  int Zehner  = 0;
  int Einer   = 0;
  int Zehntel = 0;
  int Zeichen;
  ms          = 0;
  Rx_Zeichen  = -1;

  // UART initialisieren
  setze_bit(UART_Basis + UART_Kontroll, UART_RxD_IrEn);

  // Timer initialisieren
  schreibe_bitfeld(Timer_Basis + Timer_Periode, TAKTZYKLEN_PRO_MS, 0xffffffff, 0);

  // Interrupt IP2 irgendwie freigeben
  // Interrupt IP3 irgendwie freigeben
  // Interrupt IP4 irgendwie freigeben
  // Globalen Interrupt irgendwie freigeben

  Zeit_anzeigen(Minuten, Zehner, Einer, Zehntel);
```

```
  while(1) {

    // Schleife für Betriebszustand WARTEN
    while(1) {
      Zeichen = Rx_Zeichen_holen();

      // Bei gedrücktem START-Taster oder bei Zeichen 's' Schleife WARTEN verlassen
      if('s'== Zeichen || (lese_bit(GPIO_Basis + GPIO_Eingabe) & START)) {
        break;
      }

      // Bei gedrücktem RESET-Taster oder bei Zeichen 'r' alle Werte zurücksetzen
      if('r' == Zeichen || (lese_bit(GPIO_Basis + GPIO_Eingabe) & RUECKSETZEN)) {
        Minuten = 0;
        Zehner  = 0;
        Einer   = 0;
        Zehntel = 0;
        Zeit_anzeigen(Minuten, Zehner, Einer, Zehntel);
      }
    } // Ende der Schleife WARTEN


    // Uhr starten (Timer-Interrupt freigeben)
    ms = 0;
    setze_bit(Timer_Basis + Timer_Kontroll, Timer_IrEn);

    // Schleife für Betriebszustand ZAEHLEN
    while(1) {
      if (ms >= 100) {
        loesche_bit(Timer_Basis + Timer_Kontroll, Timer_IrEn); // Timer-IR sperren
        ms = ms - 100;
        setze_bit(Timer_Basis + Timer_Kontroll, Timer_IrEn); // Timer-IR freigeben
        Zehntel++;
        if (Zehntel == 10) { Einer++;   Zehntel = 0; }
        if (Einer   == 10) { Zehner++;  Einer   = 0; }
        if (Zehner  ==  6) { Minuten++; Zehner  = 0; }
        if (Minuten == 10) {            Minuten = 0; }
        Zeit_anzeigen(Minuten, Zehner, Einer, Zehntel);
      }

      // Bei gedrücktem STOP-Taster oder bei Zeichen 'x' Schleife ZAEHLEN verlassen
      Zeichen = Rx_Zeichen_holen();
      if(Zeichen == 'x' || (lese_bit(GPIO_Basis + GPIO_Eingabe) & STOP)) {
        break;
      }
    }

    // Uhr stoppen (Timer-Interrupt sperren)
    loesche_bit(Timer_Basis + Timer_Kontroll, Timer_IrEn);

  } // Ende der Schleife ZAEHLEN
} // Ende des Hauptprogramms
```

Vorbereitung

1. Inspizieren Sie die C-ähnliche Softwarebeschreibung.
2. Machen Sie sich mit der Testbench in Beispielrechner_System_V5_A2_testbench.vhd vertraut.

Durchführung

1. Importieren Sie das Projekt „Erweiterte_Stoppuhr“ in den Workspace. Dieses enthält im Verzeichnis *source* die Assembler-Datei Erweiterte_Stoppuhr.S.
2. Ergänzen Sie im Programm die fehlenden Codeteile (mit TODO markiert). Diese können Sie aus den vorherigen Aufgaben übernehmen.
3. Ergänzen Sie das Unterprogramm Zeit_anzeigen, so dass die Zeit zusätzlich zur Anzeige auf der Siebensegmentanzeige auch auf dem UART ausgegeben wird. Dazu müssen Sie das Unterprogramm Tx_Fifo_schreiben pro auszugebendem Zeichen einmal aufrufen (siehe C-Code). Denken Sie daran, vor dem Unterprogramm-Aufruf die Caller-Saved-Register auf dem Stack zu sichern.
4. Stellen Sie sicher, dass im Programm die Werte für die beschleunigte Ausführung eingestellt sind. Hier ist die Beschleunigung nicht so hoch, um die Ausgabe per UART zu ermöglichen.
5. Lassen Sie das das Programm übersetzen. Rufen Sie nach erfolgreicher Übersetzung die Batch-Datei CopyHexFile.bat auf.
6. Übersetzen und Simulieren Sie das System mit der Stoppuhr-Software im VHDL-Simulator mit der Kommandodatei test_Beispielrechner_System_V5_A2.do. Inspizieren Sie das Wave-Fenster. Die Testbench startet und stoppt die Stoppuhr mit Zeichen über die serielle Schnittstelle. Die Uhrzeit sollte auch über die serielle Schnittstelle ausgegeben werden.
7. Stellen Sie im Programm die Werte für die Ausführung in Normalgeschwindigkeit ein. Assemblieren Sie das Programm. Starten Sie anschließend das Programm mit einer Debug-Konfiguration auf dem FPGA.
8. Starten Sie das Terminalprogramm. Überprüfen Sie, dass die Ausgabe der Stoppuhr auch im Terminalfenster angezeigt wird. Bedienen Sie die Software über Taster und über das Terminal.

Bitte legen Sie nach Abschluss des Versuchs folgende Dateien im OSCA-Dateibereich Ihrer Arbeitsgruppe in einem neuen Ordner „V5“ ab:

- Beispielrechner_System.vhd
- Beispielrechner_System.xdc
- Teste_UART.S
- Erweiterte_Stoppuhr.S