Praktikum Softprozessor SP5: Zähler und Zeitgeber

9. Juli 2014

Zusammenfassung

Die bisherige Hardware-Konfiguration wird um eine Zähler/Zeitgeber-Einheit erweitert. Diese ist in den bisherigen Rechner mit einzubauen. Anschließend sind typische Programmieraufgaben mit der Zähler/Zeitgeber-Einheit zu lösen.

1 Die Zähler/Zeitgeber-Einheit

Die in den Softprozessor einzubauende Zähler/Zeitgeber-Einheit hat zwei identische Kanäle. Jeder der beiden Kanäle $i \in \{0, 1\}$ besitzt drei über den Peripherie-Bus (PLB) les- und beschreibbare Spezialregister (Abbildung 1):

- TCSRi Kontrollregister zur Einstellung der Betriebsart und zum Lesen von Statuswerten.
- TCRi: Ein 32-Bit-Zählregister, das bei Über- oder Unterlauf das Ereignisbit im Kontroll- und Statusregister setzt.
- TLRi Ein 32-Bit-Laderegister, aus dem im Zeitgeberbetrieb das Zählregister geladen wird und in das im Zählbetrieb der Zählwert bei Aktivierung des Übernahmesignals Capt$i$ übernommen wird.

Das Kontrollwort setzt sich aus Einzelbits zusammen, die alle eine spezielle Funktion haben. Von diesen werden im weiteren nur benutzt:

PWMAi Pulsweitenmodulation: 1 – ein, 0 – aus.

Ti**INT** Ereignisbit (Interrupt-Flag): wird vom Zähler bei Über- und Unterlauf auf 1 gesetzt. Um es zu löschen, muss eine 1 in dieses Bit geschrieben werden.

ENTi Zählerfreigabe: 1 – Zählen, 0 – Zähler anhalten.

LOADi Laden bei Setzen einer 1.

ARHTi Wenn 1, Autoreload bei Überlauf / Überschreiben bei wiederholten Triggerereignis.

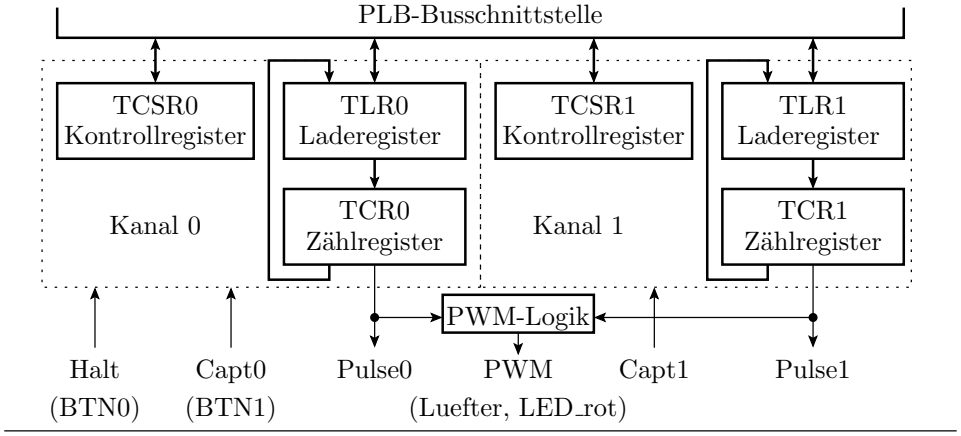
CAPTi Freigabe des exteren Übernahmesignal.

GENi Freigabe der Pulsausgabe

UDTi Zählrichtung (up/down): 0 – aufwärts, 1 – abwärts

MDTi Modus: 0 – Zeitgeber, 1 – Zeitmessung.

In die übrigen Steuerbits, die in dieser Übung nicht genutzt werden, ist bei Wertezuweisungen an des Kontrollwort der Wert 0 zu schreiben.



Aufbau der Kontrollworte TCSR*i*:

	ENALL	T*i*INT	ENIT*i*	ARHT*i*	GENT*i*	MDT*i*					
Bit 11 bis 31 ungenutzt	10	9	8	7	6	5	4	3	2	1	0*
		PWMA*i*	ENT*i*	LOAD*i*	CAPT*i*	UDT*i*					

* Bitnumerierung aus Software-Sicht. In der Hardwarebeschreibung sind die Bits genau umgekehrt nummeriert.

Abbildung 1: Zu ergänzende Zähler/Zeitgeber-Einheit

Im Header »xtmrctr_l.h« ist für jedes dieser Bits eine Maske als Konstante definiert, in der das Bit eins und die übrigen Bits null sind:

```
#define XTC_CSR_CAPTURE_MODE_MASK  0x00000001 // Zeitmessung
#define XTC_CSR_DOWN_COUNT_MASK    0x00000002 // UDT-Bit
#define XTC_CSR_EXT_GENERATE_MASK  0x00000004
#define XTC_CSR_EXT_CAPTURE_MASK   0x00000008 // CAPT-Bit
#define XTC_CSR_AUTO_RELOAD_MASK   0x00000010 // ARHT-Bit
#define XTC_CSR_LOAD_MASK          0x00000020 // LOAD-Bit
#define XTC_CSR_ENABLE_INT_MASK    0x00000040 // Zählerfreigabe
#define XTC_CSR_ENABLE_TMR_MASK    0x00000080 // Ereignisbit
#define XTC_CSR_INT_OCCURED_MASK   0x00000100 // TINT-Bit
#define XTC_CSR_ENABLE_PWM_MASK    0x00000200 // PWMA-Bit
```

Zum Schreiben und Lesen der Register sind im Header folgende Makros definiert:

```
XTmrCtr_SetControlStatusReg(BaseAddress, i, Wert) // Kontrollregister schreiben
XTmrCtr_GetControlStatusReg(BaseAddress, i)       // Kontrollregister lesen
XTmrCtr_GetTimerCounterReg(BaseAddress, i)        // Zähler lesen
XTmrCtr_GetLoadReg(BaseAddress, i)                // Laderegister lesen
XTmrCtr_SetLoadReg(BaseAddress, i, Wert)          // Laderegister schreiben
```

($i \in \{0,1\}$ – Kanalnummer). Das Zählregister kann nicht direkt beschrieben werden. Stattdessen ist der Wert in das Laderegister zu schreiben und anschließend das Load-Bit zu setzen. Für das Setzen des Load-Bits und für den Test des Ereignisbits sind im Header zwei weitere Makros definiert:

```
XTmrCtr_LoadTimerCounterReg(BaseAddress, i) // Ladebit setzen
XTmrCtr_HasEventOccurred(BaseAddress, i)    // wahr, wenn Ereignisbit gesetzt
```

Beim Setzen des Autoreload-Bits wird das Zählregister zu Beginn und bei jedem Timer-Ereignis neu geladen.

Für Zeitmessungen hat der Timer zwei Eingänge:

- Das Halt-Signal, an dem in Abbildung der Taster BTN0 angeschlossen ist, stoppt, wenn es aktiviert wird, den Zähler und
- das Übernahmesignal »Capt*i*« löst bei Aktivierung eine Übernahme des Zählregisterwertes in das Laderegister aus.

In der Betriebsart PWM (Pulsweitenmodulation) erfolgt die Ausgabe am Ausgang PWM, an den im Praktikum der Lüfter und der rote Farbkanal der Mehrfarb-LED der anzusteckenden Zusatzbaugruppe angeschlossen werden soll. Die übrigen Anschlüsse Pulse und Capt1 bleiben im Praktikum ungenutzt.

1.1 Einmalige Wartezeiten

Zur Programmierung einer Wartezeit mit einem Timer schreibt das Programm den Zählwert in das Laderegister. Dieser wird beim Aktivieren des Ladebits in das Zählregister übernommen. Dann ist das Ladebit zu deaktivieren, Abwärtszählen einzustellen und der Zähler einzuschalten. Anschließend verringert sich der Zählwert in jedem Takt um eins.[1] Bei Erreichen des Zählwertes Null wird das Ereignisbit gesetzt und der Zähler bleibt stehen. Das Programm selbst wartet in einer Schleife auf das Setzen des Ereignisbits und löscht es. Die Zähltaktfrequenz ist im Beispielentwurf 50 MHz. Der Zählwert n ist das Produkt aus der gewünschten Zeit t bis zum Zeitgeberereignis und der Zähltaktfrequenz:

$$n = t \cdot 50\,\mathrm{MHz}$$

Das nachfolgende Unterprogramm initialisiert den Timer. Die erste Anweisung schreibt den Zählwert n in das Laderegister. Die zweite Anweisung setzt das Ladebit, so das der Zähler den Wert aus dem Laderegister übernimmt. Die dritte Anweisung schaltet den Zähler ein, die Zählrichtung auf abwärts und löscht das Ereignisbit. Das Statuswort dafür wird durch ODER-Verknüpfung der Maskenwerte der zu setzenden Bits, die in »xtmrctr_l.h« definiert sind, gebildet. In diesen Maskenwerten ist das im Namen enthaltene Statusbit jeweils eins und die restlichen Bits sind null.

```
void initTimer0(int n){
 XTmrCtr_SetLoadReg(XPAR_XPS_TIMER_0_BASEADDR, 0, n);          // TLR0 laden
 XTmrCtr_LoadTimerCounterReg(XPAR_XPS_TIMER_0_BASEADDR, 0); // LOAD-Bit setzen
 XTmrCtr_SetControlStatusReg(XPAR_XPS_TIMER_0_BASEADDR, 0,
              XTC_CSR_DOWN_COUNT_MASK                          // abwärtszählen
            | XTC_CSR_ENABLE_TMR_MASK                          // Zähler ein
            | XTC_CSR_INT_OCCURED_MASK);                       // Ereignisb. löschen
}
```

Das nachfolgende Unterprogramm wartet, bis der Zähler null ist. Dazu liest es in einer Schleife im Unterprogramm »XTmrCtr_HasEventOccurred()« das Statuswort und prüft, ob darin das Ereignisbit gesetzt ist.

```
void waitTimer0(){while (!XTmrCtr_HasEventOccurred(XPAR_XPS_TIMER_0_BASEADDR, 0));
```

Um eine Wartezeit in einem Programm einzufügen, sind beide Unterprogramme nacheinander aufzurufen. Alternativ können sie auch zu einem Unterprogramm zusammengefasst werden:

```
void sleep(int n){ // n = t*50 MHz  -- Zählwert für die Wartezeit
 initTimer0(n);
 waitTimer0();
}
```

Das nachfolgende Beispielprogramm erlaubt eine Einstellung der Blinkfrequenz einer Leuchtdiode mit einem Schalter. Das Hauptprogramm gibt in einer Endlosschleife den Wert von y aus, liest den Schalterwert in x ein, wartet für eine Zeit

$$t_\mathrm{w} = (s+1) \cdot 10\,\mathrm{ms}$$

(s – Schalterwert) und invertiert Bit 0 von y:

[1] Aufwärtszählen ist auch möglich. Dann wird das Ereignisbit bei Überlauf gesetzt und die Anzahl der Zählpulse ist die Differenz aus dem Überlauf- und dem Reload-Wert.

```
int main(){
  int x, y=0;
  while (1){
    XGpio_WriteReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR, y);
    x = XGpio_ReadReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR);
    sleep ((x+1)*500000);
    y ^= 1;
  }
}
```

1.2 Periodische Zeitgeberereignisse und Systemuhr

Zur Erzeugung periodischer Zeitereignisse wird zusätzlich das Autoreload-Bit ARHT0 im Statusregister gesetzt. Es bewirkt, dass der Zähler zu Beginn und bei Erreichen des Zustands null wieder neu aus dem Laderegister initialisiert wird und weiterzählt. Ein Anwendungsbeispiel für den Autoreload-Modus ist die Programmierung einer Systemuhr. Diese soll die Zeit in Sekunden seit Systemstart zählt. Das erfordert ein Zeitereignis alle 50.000.000 Zählschritte.

Das Hauptprogramm auf einem Mikrorechner besteht oft aus einer Ereignisschleife, in der zyklisch nacheinander alle Ereignisbits abgefragt werden. Wenn sie gesetzt sind, wird der zugeordnete Programmbaustein ausführt, sonst wird er übersprungen. Typische Ereignisse sind der Empfang oder das Versenden eines Bytes über die serielle Schnittstelle oder wie hier das periodische Timer-Ereignis der Systemuhr.

Das nachfolgende Programmbeispiel kombiniert zwei Tasks:

- Die serielle Ausgabe eines Zeichens aus einem Schreibpuffer, wenn der Schreibpuffer noch Zeichen enthält und die serielle Schnittstelle ein neues Zeichen aufnehmen kann und
- eine Uhrenfunktion, die immer nach einer Sekunde eine Zeichenkette in den Schreibpuffer schreibt.

Ein Schreibpuffer besteht aus einem Feld einer bestimmten Größe und zwei Zeigern. Der Zeiger »wptr« zeigt auf den nächsten freien Platz zum Schreiben und der Zeiger »sptr« auf das nächste zu versendende Zeichen. Die zusätzlichen Statuswerte »voll« und »leer« signalisieren, ob noch freier Platz zum beschreiben bzw. noch Zeichen zum versenden verfügbar sind. Für den Puffer ist eine Schreibfunktion für ein Zeichen in den Puffer und auf diese aufsetzend eine Schreibfunktion für eine Zeichenkette in den Puffer definiert.

Das Hauptprogramm initialisiert das Lade- und das Statusregister und fragt in einer Endlosschleife ab, ob eine der beiden Aufgaben ausgeführt werden soll. Wenn ein Zeichen zu versenden ist, schreibt es das Programm in das Senderegister, schaltet den Zeiger zirkular weiter und setzt, falls danach kein Zeichen mehr im Puffer steht, das Flag »leer«. Wenn das Ereignisbit des Timers gesetzt ist, wird ein Text in den Puffer geschrieben und das Ereignisbit durch Neuzuweisung des Statuswortes, in dem das Ereignisbit gesetzt ist, gelöscht.

```
#include "xparameters.h"
#include "xgpio_l.h"
#include "xuartlite_l.h"
#include "xtmrctr_l.h"

// Datenstruktur für einen Sendepuffer
#define SBufSize 16
char SendBuff[SBufSize];
char * wptr=SendBuff, * sptr=SendBuff;
int voll=0, leer=1;

// Schreiben eines Zeichens in den Sendepuffer
int putc(char c){
  leer = 0;
  if (voll==1) return 1;
  *wptr = c;
```

```
  wptr++;
  if (wptr == SendBuff+SBufSize) wptr = SendBuff;
  if (wptr == sptr) voll=1;
  return 0;
}

// Schreiben einer Zeichenkette in den Sendepuffer
void SendStr(char * s){
        while (!(*s==0)&&(putc(*s)==0)) s++;
}

int main(){
  int StatusCnf = XTC_CSR_DOWN_COUNT_MASK    // abwärtszählen
                | XTC_CSR_ENABLE_TMR_MASK    // Zähler ein
                | XTC_CSR_INT_OCCURED_MASK   // Ereignisflag. löschen
                | XTC_CSR_AUTO_RELOAD_MASK; // Autoreload ein

  // Lade- und Kontrollregister des Timers initialisieren
  XTmrCtr_SetLoadReg(XPAR_XPS_TIMER_0_BASEADDR, 0, 50000000);
  XTmrCtr_SetControlStatusReg(XPAR_XPS_TIMER_0_BASEADDR, 0, StatusCnf);
  while (1){

  // Wenn der Sende-FIFO nicht voll und der Sende-Buffer nicht leer ist
    if (!XUartLite_IsTransmitFull(XPAR_UART_BASEADDR) && !leer){
         // Sende ein Zeichen und schalte des Schreibzeiger zirkular weiter
          XUartLite_WriteReg(XPAR_UART_BASEADDR, XUL_TX_FIFO_OFFSET, *sptr);
        voll = 0;
         sptr++;
          if (sptr==SendBuff+SBufSize) sptr = SendBuff;
         if (sptr==wptr) leer=1;
        }

   // Wenn der Zeitgeber abgelaufen ist (jeweils nach 1 Sekunde)
   if (XTmrCtr_HasEventOccurred(XPAR_XPS_TIMER_0_BASEADDR, 0)) {
     // Text in den Puffer schreiben
     SendStr("Sekunde um\n");
     // Ereignisflag des Timers löschen
     XTmrCtr_SetControlStatusReg(XPAR_XPS_TIMER_0_BASEADDR, 0, StatusCnf);
    }
  }
}
```

1.3 Zeitmessung

Im Zeitmessmodus sind im Konfigurationsregister folgende Bits zu setzen:

- MDT: Aktivierung der Betriebsart »Zeit messen«
- ENT: Zähler einschalten
- CAPT: Freigabe des externen Übernahmesignals und
- TINT: Ereignisbit löschen.

Der Zähler startet. Bei einer steigenden Flanke am Übernahmeeingang (BTN0 wird gedrückt), wird der Zählerwert in das Laderegister übernommen und das Ereignisbit gesetzt. Ein aktives Halt-Signal (BTN1 gedrückt) hält den Zähler an. Das nachfolgende Beispielprogramm führt die Initialisierung aus und liest in einer Endlosschleife, immer wenn das Ereignisbit gesetzt ist, den in das Laderegister kopierten Wert als Dezimalzahl über die serielle Schnittstelle an den PC und löscht das Ereignisbit durch Neuschreiben des Konfigurationswertes. Zum Empfang der Programmausgaben ist das Monitorprogramm für die serielle Schnittstelle auf dem PC zu starten.

```
// Ausgabe einer positiven ganzen Zahl als Ziffernfolge
void term_write_unsigned(unsigned z){
```

```
  unsigned x, y;
  for (x = 1, y = z;y >= 10; x *= 10, y /= 10);
  while (x > 0){
    char c = z / x;
    XUartLite_SendByte(XPAR_UART_BASEADDR, ('0' + c));
    z -= c * x;
    x /= 10;
  }
}

int main() {
  int i = 0;
  // Timer konfigurieren
  int StatusCnf = XTC_CSR_CAPTURE_MODE_MASK // Zeitmessung
                | XTC_CSR_ENABLE_TMR_MASK   // Timer einschalten
                | XTC_CSR_EXT_CAPTURE_MASK  // Externe Capture-Signale aktivieren
                | XTC_CSR_INT_OCCURED_MASK; // Timer event löschen

  // Einstellungen am Timer vornehmen
  XTmrCtr_SetControlStatusReg(XPAR_XPS_TIMER_0_BASEADDR, 0, StatusCnf);
  while (1){
    if (XTmrCtr_HasEventOccurred(XPAR_XPS_TIMER_0_BASEADDR, 0))                    {
        int count = XTmrCtr_GetLoadReg(XPAR_XPS_TIMER_0_BASEADDR, 0);
      term_write_unsigned(count);
      XUartLite_SendByte(XPAR_UART_BASEADDR, '\n');
      XTmrCtr_SetControlStatusReg(XPAR_XPS_TIMER_0_BASEADDR, 0, StatusCnf);
   }
  }
}
```

1.4 PWM-Steuerung

Pulsweitenmodulation (PWM) dient zur stufenlosen Leistungssteuerung. Am Ausgabeelement, z.B. einer Motorwicklung oder einem Lastwiderstand, wird die Spannung ein- und ausgeschaltet (Abbildung 2). Die im Ausgabeelement umgesetzte Leistung beträgt im Mittel

$$\bar{P}_L = \frac{t_{\text{ein}}}{T_{\text{P}}} \cdot P_{\max}$$

(t_{ein} – Einschaltzeit; T_{P} – Periodendauer; $P_{\max}$ – Leistungsumsatz bei Dauereinschaltung). Zur Erzeugung eines PWM-Signals werden beide Zeitgeberkanäle der Zähler/Zeitgeber-Einheit genutzt. Der Ladewert von Kanal 0 legt die Periodendauer und der Ladewert von Kanal 1 die Einschaltdauer fest. Zur PWM-Signalerzeugung sind beide Laderegister zu beschreiben und für beide Kanäle die Flags zum Einschalten des Zählers, für Abwärtszählen, zur PWM-Aktivierung und zur Freischaltung des PWM-Ausgangs zu aktivieren. Im nachfolgenden Beispielprogramm folgt nach der Initialisierung eine Endlosschleife, die nichts tut. Das PWM-Signal wird unabhängig vom Programmablauf erzeugt. Es hat eine Periodendauer $T_{\text{P}} = 1\,\text{s}$ und einer Einschaltzeit von $t_{\text{ein}} = 0{,}5\,\text{s}$.

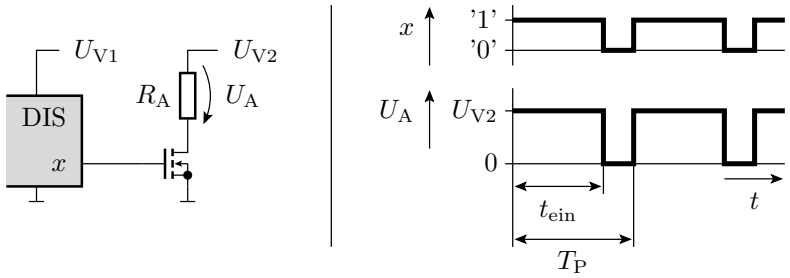


Abbildung 2: Prinzip der Pulsweitenmodulation

```
// Projekt: PWM, Programm: main_PWM.c
int main() {
        // Kontrollwort zusammensetzen
        u32 timerCfg = XTC_CSR_DOWN_COUNT_MASK    // abwärts zählen
                     | XTC_CSR_ENABLE_TMR_MASK    // Timer einschalten
                     | XTC_CSR_ENABLE_PWM_MASK    // PWM einschalten
                     | XTC_CSR_EXT_GENERATE_MASK; // PWM-Ausgang aktivieren

        // Für beide Timer Ladewert einstellen
        XTmrCtr_SetLoadReg(XPAR_XPS_TIMER_0_BASEADDR, 0, 50000000);
        XTmrCtr_SetLoadReg(XPAR_XPS_TIMER_0_BASEADDR, 1, 25000000);
        // In beide Timer Kontrollwort schreiben
        XTmrCtr_SetControlStatusReg(XPAR_XPS_TIMER_0_BASEADDR, 0, timerCfg);
        XTmrCtr_SetControlStatusReg(XPAR_XPS_TIMER_0_BASEADDR, 1, timerCfg);
        while (1); // Endlosschleife
}
```

Zur Visualisierung der PWM-Ausgabe ist an Port A1 über den zugehörigen Adapter die Zusatzbaugruppe mit dem Lüfter und der Mehrfarb-LED anzuschließen. Das Programm gibt das PWM-Signal auf dem Rot-Kanal der Mehrfarbleuchtdiode und, wenn an die Adaperbaugruppe zusätzlich eine 12V-Versorgungsspannung angeschlossen ist, auf den Lüfter aus.

2 Hardware-Entwurf

Abbildung 3 zeigt das zu entwerfende Gesamtsystem. Das System vom ersten Praktikum aus Prozessor, Speicher, parallelen Schnittstellen, UART und Debugschnittstelle wird komplett übernommen. Zusätzlich wird an den Peripheriebus eine Zähler/Zeigebereinheit angeschlossen. Die Eingänge »Halt« und »Cap« zum Anhalten des Zählers von Kanal null und »Cap« zur Steuerung der Übernahme des Zählwertes in der Betriebsart Zeitmessung sind mit Tastern auf der Versuchsbaugruooe zu verbinden und der PWM-Ausgang mit dem Steuereingang für den Lüfter. Um den Lüfter anzusteuern sind die Zusatzbaugruppe mit dem Lüfter an den Erweiterungsstecker A1 der Versuchsbaugruppe zu stecken und an die Zusatzbaugruppe zusätzlich die 12V-Versorgungsspannung anzuschließen.den

Kopieren Sie sich aus den bisherigen Projekten folgende Entwurfsdateien in ein neu angelegtes Unterverzeichnis

```
.../Aufg5/data/system.ucf
.../Aufg5/system.xmp
.../Aufg5/system.mhs
.../Aufg5/system.mss
```

Starten Sie das Entwurfsprogramm:

`Anwendungen ▷ Umgebung ▷ Xilinx Platform Studio (EDK)`

Öffnen Sie das Projekt

`File ▷ Open Projekt ▷ <Auswahl von .../Aufg5/system.xmp>`

Alternativ können Sie auch den kompletten Entwurfsablauf vom ersten Aufgabenblatt wiederholen. Ergänzen Sie aus dem IP-Katalog wie in Abbildung 4 einen XPS Timer/Counter, benennen Sie ihn in »Timer« um und schließen Sie ihn wie Abbildung 4 an den PLB-Bus an.

Als nächstes sollen in der Ansicht »System Assembly View, Ports« folgende Anschlüsse der Zähler-/Zeitgeber-Einheit mit externen Anschlüssen der Baugruppe verbunden werden (Abbildung 5):

- Das Halt-Signal mit der Taste BTN0 auf der Baugruppe
- Das Capture-Signal von Timer 0 mit der Taste BTN1 und
- der PWM-Ausgang mit dem Lüfter und dem Rot-Anschluss der Mehrfarb-LED auf der an Port A1 anzusteckenden Zusatzbaugruppe.

Zusätzlich sind in der UCF-Datei folgende Zuordnungen zu ergänzen:

```
NET "BTN0"    LOC = "M13";
NET "BTN1"    LOC = "M14";
NET "Luefter" LOC = "B1";
NET "LED_rot" LOC = "C1":
```

Der Zähler/Zeitgeber-Einheit ist des weiteren in der Ansicht »System Assembly View, Addresses« mit »Generate Addresses« ein Adressbereich zuzuordnen (Abbildung 6). Abschließend ist mit

```
Project ▷ Export Hardware Design to SDK ▷ Export Only
```

die Schaltung des Prozessors zu synthetisieren, die Header mit den Hardware-Adressen zu generieren etc. und alle Daten für SDK bereitzustellen.

3 Software-Entwurf

SDK unter Linux Starten:

```
Anwendungen ▷ Umgebung ▷ Xilinx Platform Studio (SDK)
```

Den Arbeitsbereich des Projekts auswählen:

```
.../Prakt_Softprozessor/Aufg5/SDK/SDK_Workspace
```

Die Hardware-Spezifikationsdatei, die angefordert wird, ist:

```
.../Prakt_Softprozessor/Aufg5/SDK/SDK_Export/hw/system.xml
```

Erzeugen der Software-Plattform (generieren der Header und Treiber):

- »File« ▷ »New« ▷ »Project« ▷ »Software Platform«
- Projektname »Timerprojekt«
- fertigstellen.

Nach der Software-Plattform ist ein Projekt anzulegen:

- »File« ▷ »New« ▷ »New Managed Make C Application Project« ▷ Namen »Blinker« festlegen, »Empty Application« einstellen ▷ durchklicken bis Fertigstellen

In dem Projekt ist wiederum eine Programmdatei anzulegen:

- »File« ▷ »New« ▷ »Source File« ▷ Namen mit Endung ».c« festlegen und fertigstellen

Das nachfolgende C-Programm zum Test der Hardware ist um die in Abschnitt 1.1 beschriebenen Unterprogramme »initTimer0«, »waitTimer0« und »sleep« zu ergänzen und lässt die niederwertigste Leuchtdiode auf der Baugruppe blinken.

```
#include "xparameters.h"
#include "xgpio_l.h"
#include "xuartlite_l.h"
#include "xtmrctr_l.h"

... // Definition der drei Unterprogramme

int main(){
 unsigned char x=0;
 while (1){
   XGpio_WriteReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR, 0, x);
   sleep (20000000);
   x ^= 1;
 }
}
```

Das Programm wird beim Abspeichern wieder automatisch übersetzt. Nach Beseitigung aller Fehler mit

- Run ▷ Run...
- Projekt »Blinker« und C/C++ Applikation »Debug/Blinker.elf« auswählen.
- Run

starten. Ein Test im Schrittbetrieb ist nur begrenzt möglich, weil auch im Schrittbetrieb der Zähler im Timer mit dem normalen Systemtakt weiterzählt. Start nach Programmänderung durch Klicken auf das grün eingerahmte Dreieck.

4 Aufgaben

Aufgabe 4.1: Erzeugen einer periodischen Blinkfolge

1. Führen Sie den Hardware- und den Software-Entwurf aus den Abschnitten 2 und 3 zu Ende und testen Sie das Beispielprogramm aus Abschnitt 1.1 (Programm main_b1_1.c von der Web-Seite).
2. Erweitern Sie das Beispielprogramm aus der Voraufgabe so, dass es zyklisch auf der Leuchtdiode LD0 eine Blinkfolge mit der Einschaltzeit »lang-kurz-kurz-lang-kurz-lang-lang-kurz-Pause« ausgibt. Die Einschaltzeit für »kurz« sei etwa eine halbe und für »lang« 2 Sekunden. Die nomale Ausschaltzeit zwischen zwei Blinkzeichen sei eine halbe und für die Pause 3 Sekunden.

Aufgabe 4.2: Uhr mit Monitorausgabe

1. Legen Sie eines neues Projekt »b1_2_Uhr« an und vervollständigen und testen Sie das Uhrenprogramm aus Abschnitt 1.2 (Programm main_b1_2.c von der Web-Seite).
2. Schreiben Sie ein Unterprogramm, das eine Sekundenangabe in eine Zeichenkette der Form

   $$mm : ss\backslash n$$

   (mm und ss – jeweils Zehner und Einer der Minuten und Sekunden) umwandelt und in eine Zeichenkettenvariable ablegt, sowie einen Testrahmen für das Unterprogramm. Testen Sie das Unterprogramm mit dem Debugger.
3. Erweiteren Sie das vorgegebene Uhrenprogramm aus Aufgabenteil 1 so, dass bei jedem Zeitereignis der auf der Leuchtdiode LD0 ausgegebene Wert invertiert wird.
4. Erweiteren Sie das vorgegebene Uhrenprogramm aus Aufgabenteil 3 weiterhin so, dass die Zeitangabe bei jedem Zeitereignis (aller Sekunden) in den Puffer geschrieben und von dem anderen Task über die serielle Schnittstelle an den PC gesendet wird.

Aufgabe 4.3: Zeitmessung

1. Legen Sie ein neues Projekt Zeitmessung an und vervollständigen und testen Sie darin das Beispielprogramm aus Abschnitt 1.3.

2. Ändern Sie das Beispielprogramm aus Abschnitt 1.3 so ab, dass die Tastendrücke von BTN1 gezählt, die Zeitdifferenz zur vorherigen Tastenbetätigung bestimmt und beide Werte als Ausgabezeile wie folgt über die serielle Schnittstelle an den PC gesendet werden:

   ```
   1:  0,128s
   2: 32,127s
   ....
   ```

   (Erste Zahl fortlaufende Nummer der Tastenbetätigung, zweite Zahl Zeitdifferenz in Sekunden mit drei Nachkommastellen. Rechtsbündige Ausrichtung beider Zahlenwerte untereinander. Leerzeichen für fehlende führende Ziffern.)

Aufgabe 4.4: PWM-Ausgabe

1. Legen Sie ein neues Projekt PWM an und vervollständigen und testen Sie darin das PWM-Program aus Abschnitt 1.4.

2. In der Hauptschleife, die in dem vorgegebenen PWM-Programm in Abschnitt 1.4 leer ist, soll ein Programmteil ergänzt werden, der von der seriellen Schnittstelle auf eine Zifferenfolge:

   ```
   Ziffer,Ziffer <Enter>
   ```

   wartet, diese, nach einem korrekten Empfang aller drei Zeichen in eine positive ganze Zahl umrechnet, 500000 multipliziert und als neuen Pulsbreitenwert in das Laderegister von Kanal 1 schreibt.

3. Ändern Sie die Werte, die in die Laderegister für die Periodendauer und die relative Pulsbreite geschrieben werden so, dass die Periodendauer auf 50µs reduziert und der über die serielle Schnittstelle übergebene Zahlenwert die eingestellte relative Pulsbreite in Prozent ist.

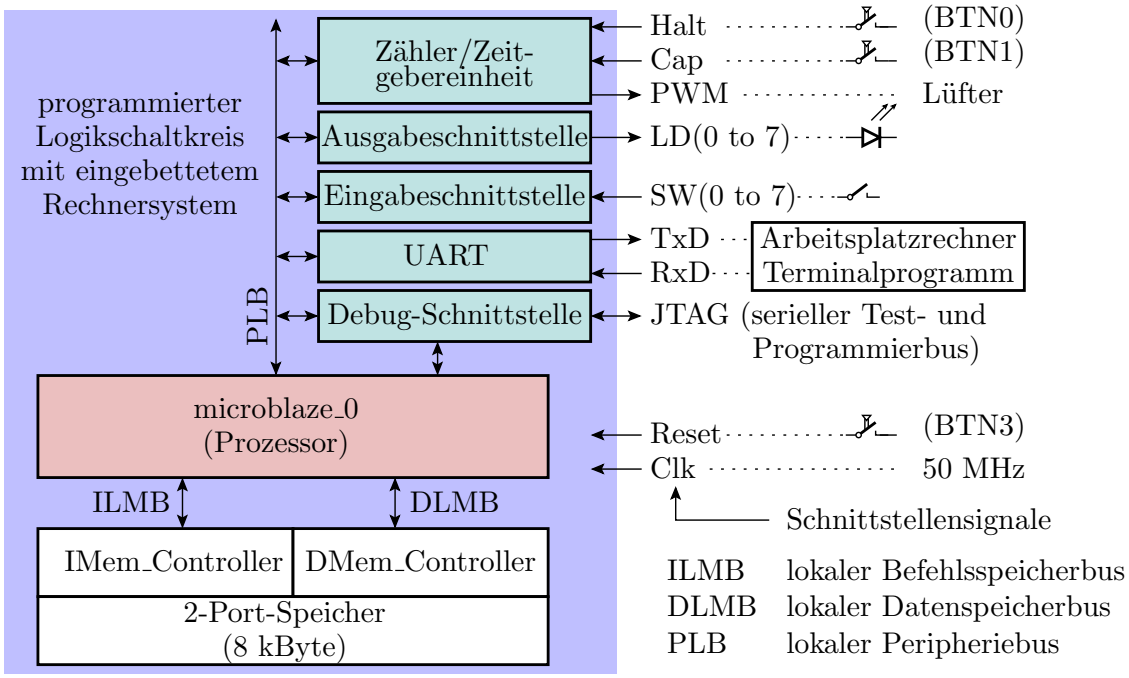


Abbildung 3: Zu entwerfendes Rechnersystem

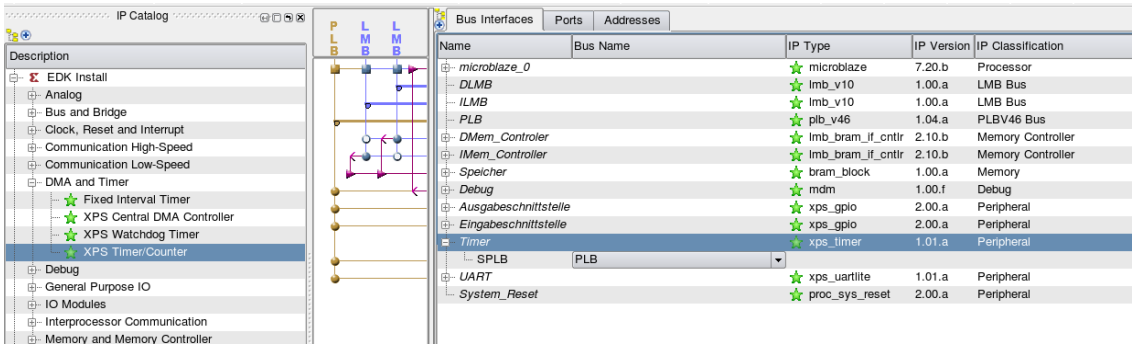


Abbildung 4: XPS Timer/Counter IP Core hinzugefügt und an den PLB angeschlossen

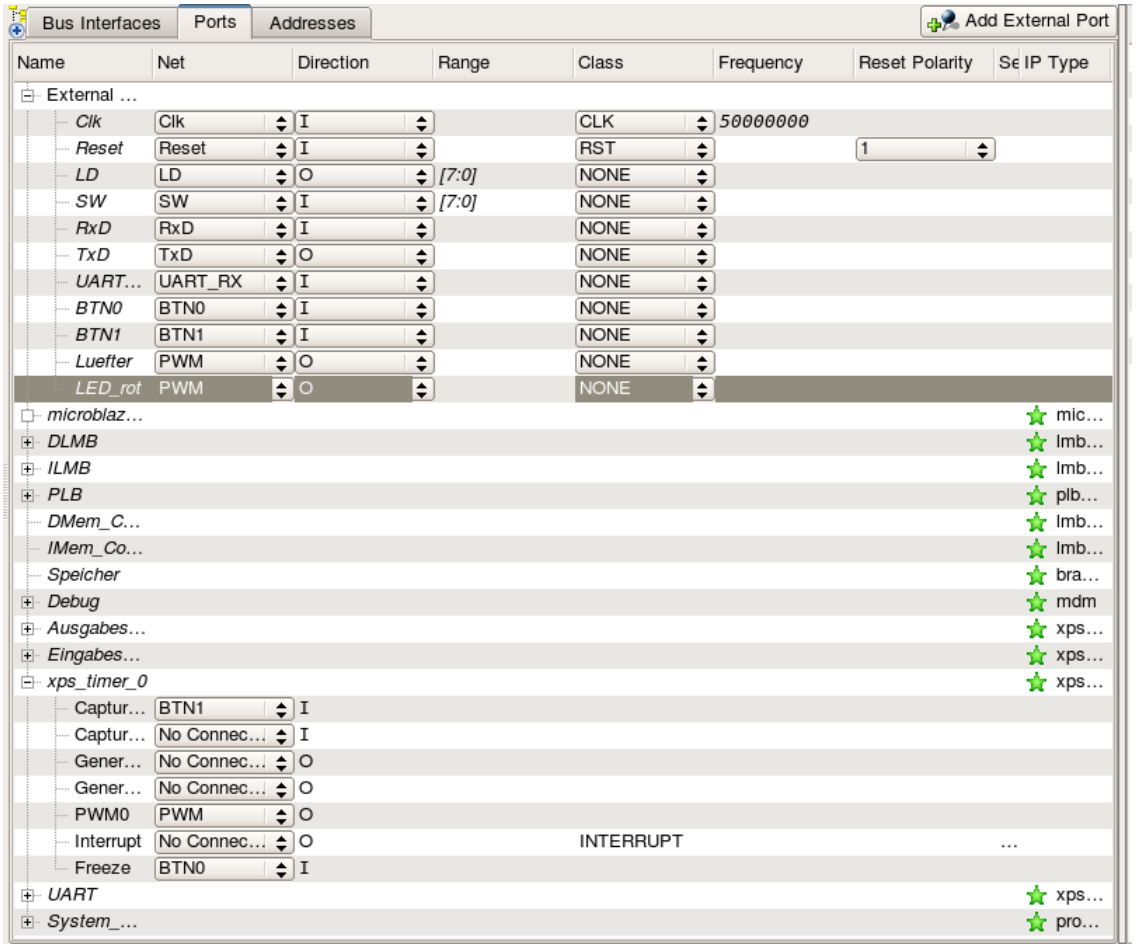


Abbildung 5: Anschlusszuordnung

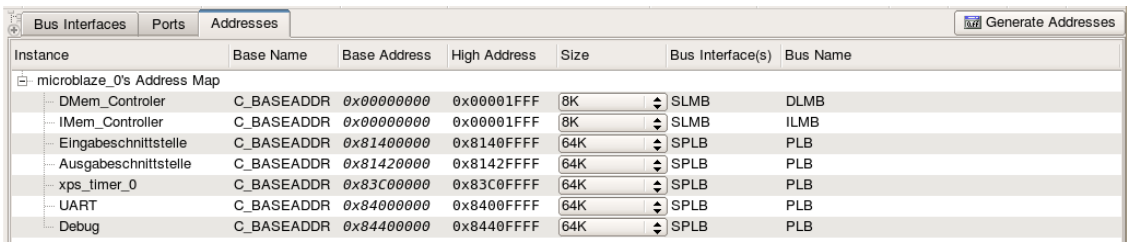


Abbildung 6: Anschlusszuordnung