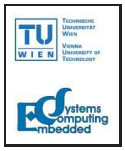

Martin Delvai

SPEAR2: Erweiterungskonzept

10. April 2007

Zusammenfassung

Spear2 Prozessorkern kann durch verschiedene System Module, Extension Module und AMBA Module an unterschiedlichen Bedürftnisse angepasst werden. Alle Module werden in den Datenspeicher gemapped und können daher mittels normaler Load/Store Operationen angesprochen werden. In diesem Dokument wird gezeigt, wie und wo die Module im Datenspeicher gemapped werden.

Inhaltsverzeichnis

1 Einleitung

Der Spear2 kann auf folgende Weise erweitert werden:

- Aktivieren interner System Module (z.B. Debug Unit)
- Hinzufügen von sogenannten Extension Modulen
- Hinzufügen von AMBA Modulen

Alle Module werden in den Datenspeicher gemapped. Dadurch kann auf die Module mit einfachen Load/Store Befehlen zugegriffen werden.

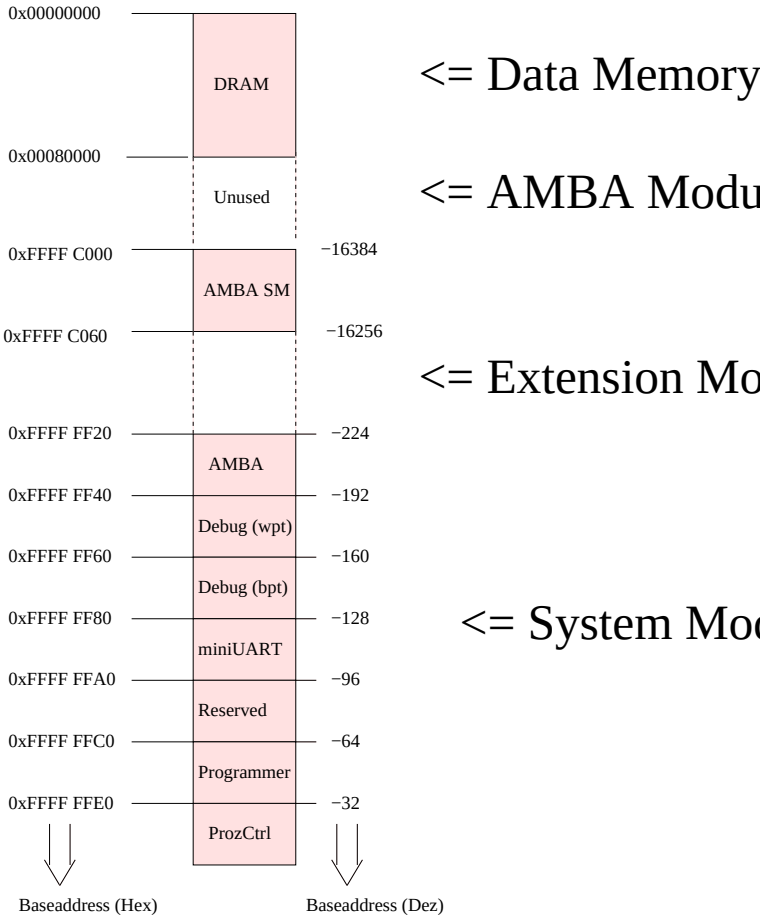

Abbildung 1: Memory Mapping

2 Interne System Module

Interne System Module haben denselben Aufbau wie Extension Module. Die System Module sind jedoch nicht nach aussen sichtbar und können mittels Konstanten im Spear2 Konfigurationfile aktiviert werden.

Er gibt folgende System Module

- Prozessor Control Modul [1]: Dieses Modul enthält das Prozessor Status Register, den Interrupt Handler und die Framepointer Register und stellt somit ein Kernelement des Prozessors dar. Das Prozessor Control Modul ist somit ein integraler Bestandteil des Prozessorkerns und kann nicht deaktiviert werden.

- Programmer Modul: Dieses Modul wird für den Downloaden eines Programms in den Befehlsspeicher des Prozessorkern verwendet: Das Programm wird über eine Kommunikationsschnittstelle (z.B. serielle Schnittstelle) vom PC in dieses Modul geladen. Anschließend programmiert das Modul den Befehlsspeicher.

- RS232 oder miniUART Modul: Diese Kommunikationsschnittstelle kann entweder für den Programmdownload oder als Debug Interface für den GDB verwendet werden.

- Debug Suport Unit (GBD-Interface): Standard Debug Interface für den GDB.

- AMBA Interface Modul: Durch Aktivieren dieses Moduls wird der Prozessorkern durch ein AMBA Interface erweitert. Es werden dabei das AHB als auch das APB Interface direk nach außen geführt.

- AMBA Shared Memory Modul: Dabei handelte es sich um einen Speicher, auf den sowohl das AMBA Modul als auch der Prozessorkern zugreifen können. Daten, die in ein AMBA Modul geschrieben bzw. von einem AMBA Modul gelesen wurde, werden in diesem Speicher abgelegt.

Diese System Module sind in den obersten Bereich des Datenspeichers gemappt, ihre Position wird durch die “Baseaddress” definiert. Diese Module werden wie normale Speicherstellen im Datenspeicher angesprochen , d.h. sie können byte-, half-word- und word-weise gelesen und beschrieben werden. Im Folgenden wird die Registerbelegung zweier Module (Prozessor Control Unit und miniUART Modul) exemplarisch angeführt.

[1] Auf Grund einer inkonsistenten Nomenklatur wird dieses Modul oft auch als System Control Modul bezeichnet

ProcCtrl Register − 32 Bit Version

CONFIG_C (−29)	CONFIG (−30)	STATUS_C (−31)	STATUS (−32)	← Baseaddress
INT_MASK_H (−25)	INT_MASK_L (−26)	INT_PROT_H (−27)	INT_PROT_L (−28)	
FPTR_W_3(−21)	FPTR_W_2(−22)	FPTR_W_1(−23)	FPTR_W_0(−24)	
FPTR_X_3(−17)	FPTR_X_2(−18)	FPTR_X_1(−19)	FPTR_X_0(−20)	
FPTR_Y_3(−13)	FPTR_Y_2(−14)	FPTR_Y_1(−15)	FPTR_Y_0(−16)	
FPTR_Z_3(−9)	FPTR_Z_2(−10)	FPTR_Z_1(−11)	FPTR_Z_0(−12)	

Abbildung 2: Processor Control Modul Register

Eine genaue Beschreibung des Prozessor Control Moduls ist in [1] zu finden.

miniUART Register − 32 Bit Version

CONFIG_C (−125)	CONFIG (−126)	STATUS_C (−127)	STATUS (−128)	← Baseaddress
MSG_H (−121)	MSG_L (−122)	UBRS_H(−123)	UBRS_L(−124)	
Unused	Unused	Unused	UART_CMD (−120)	
Unused	Unused	Unused	Unused	
Unused	Unused	Unused	Unused	
Unused	Unused	Unused	Unused	

UART_CMD (−120)

ErrI	EI		A	S	A		E V	S
7	6	5		4	3	2	1	0

CONFIG_C (−125)

ParEna	ParOdd	Stop	TrCtrl		M S G − L E N G T H			
7	6	5	4	3	2	1	0	

Abbildung 3: miniUART Modul Register

Eine genaue Beschreibung des miniUART ist in [3] zu finden. Die Debug Unit ist für den Benutzer transparent, eine Beschreibung des AMBA Interfaces ist in [2] zu finden.

3 Extension Module

Die Extension Module werden wie normale Speicherstellen im Datenspeicher angesprochen, d.h. sie können byte-, half-word- und word-weise gelesen und beschrieben werden. Im Folgenden wird am Beispiel des Display Modules

der Aufbau und die Art und Weise wie diese Module angesprochen werden können beschrieben.

Display/LED – 32 Bit Version

CONFIG_C (−93)	CONFIG (−94)	STATUS_C (−95)	STATUS (−96) ← Baseaddress
LED (−88)	DISPLAY (−90)	PRESC_H (−91)	PRESC_L (−92)
Unused	Unused	Unused	Unused
Unused	Unused	Unused	Unused
Unused	Unused	Unused	Unused
Unused	Unused	Unused	Unused
Unused	Unused	Unused	Unused
Unused	Unused	Unused	Unused

Abbildung 4: Display Modul Register

Das Display Modul kann eine 7 Segment Anzeige mit 2 Digits und 8 LEDs ansteuern. Die Daten die auf die 7 Segment Anzeige ausgegeben werden sollen müssen ins DISPLAY Register geschrieben werden. Die LEDs werden übers LED Register angesprochen. Bei den LED ist zu beachten, dass diese *low-active* sind, d.h. will man eine LED zum Leuchten bringen, so muss man das entsprechende Bit auf '0' setzen. Die Digits werden im Zeitmuliplex angesprochen - die PRESC_L und PRESC_H Register definieren die Frequenz mit der die Digits angesprochen werden. Diese Register werden nach dem Reset defaultmäßig auf *FF* gesetzt und müssen für die Verwendung des Display nicht umgesetzt werden. Die Register *Status_C* und *Config_C* werden im Display Modul nicht benutzt.

3.1 Beispiel Programm: Display Ansteuerung

Das Testprogramm soll einfach einen Index-Wert am Display ausgeben. Im Folgenden ist das Listing des Programms zu sehen.

Listing 1: Display Ansteuerung in C

```
1  #include <stdint.h>

   #define Dis7Seg_BASE (0xFFFFFEE0)
   #define Dis7Seg_Status (*(volatile uint16_t *const) (Dis7Seg_BASE))
5  #define Dis7Seg_DisData (*(volatile uint8_t *const) (Dis7Seg_BASE+6))
   #define Dis7Seg_LedData (*(volatile uint8_t *const) (Dis7Seg_BASE+7))

   int main(int argc, char **argv)
   {
10   int i;
```

```
  char dis_data;
  char led_data;


  while(1) {
    for (i = 0; (i < 256); i++)
    {
      long long counter;
      for (counter = 0; (counter < 200000); counter++)
        {
          asm("nop");
        }
      Dis7Seg_DisData =i;
 }
}
```

Im ersten Teil des Programms werden Pointer auf die Register des Display Moduls gesetzt. Dabei ist es zwecks einfacherer Wartbarkeit von Vorteil die Baseadresse zu definierten und daraus die Adressen der weiteren Register des Moduls abzuleiten.

Das Hauptprogramm durchläuft zwei Schleifen, wobei die innere Schleife lediglich eine Verzögerung für die Ausgabe implementiert.

4 AMBA Module

Das AMBA Interface [2] ist mit der GRLIB (www.gaisler.com), einer Sammlung von (AMBA-) IP Modulen, kompatibel. Es können somit beliebige AMBA Module aus der GRLIB verwendet und an Spear2 angeschlossen werden können.

Auf den AMBA Bus werden Daten und Adressen gelegt, wobei jedem Modul eine eindeutige Adresse zugeordnet ist. Damit Spear2 einen AMBA Transfer durchführen kann müssen somit folgende Schritte durchgeführt werden:

1. Wenn ein Schreibzugriff erfolgen soll so müssen die Daten ins Shared Memory gelegt werden.

2. AMBA Modul muss konfiguriert, d.h. die Zieladdresse des AMBA Models als auch die Position der Daten im Shared Memory müssen eingegeben werden. Wenn ein Lesezugriff erfolgt, so werden die gelesenen Daten an die entsprechende Stelle im Shared Memory abgelegt.

3. Read oder Write Command muss im AMBA Modul gesetzt werden.

Dieser Aufwand ist für größere Datentransfers gerechtfertigt, jedoch extrem ineffizient für einfache Zugriffe (z.B. UART Zugriff). Daher wurde auch ein “transparente” Zugriffsmodus implementiert. Im diesem Modus werden die internen Lese- und Schreibzugriffe direkt auf den AMBA Bus weitergleitet.

An dieser Stelle wird das Testprogramm für das SPI Interface Module angeführt. Dabei handelt es sich um eine AMBA APB Module, dass direkt – also mittels Transparent Mode – angesprochen wird.

Listing 2: Display Ansteuerung in C

```
#define AMBA_BASE (0xFFFFFF00)
#define AMBA_STATUS (*(volatile char *const) (AMBA_BASE))
#define AMBA_STATUS_C (*(volatile char *const) (AMBA_BASE+1))
#define AMBA_CONFIG (*(volatile char *const) (AMBA_BASE+2))
#define AMBA_CONFIG_C (*(volatile char *const) (AMBA_BASE+3))
#define AMBA_SLOT1_CONFIG (*(volatile char *const) (AMBA_BASE+4))
#define AMBA_SLOT1_MEM_OFFSET (*(volatile char *const) (AMBA_BASE+5))
#define AMBA_SLOT2_CONFIG (*(volatile char *const) (AMBA_BASE+6))
#define AMBA_SLOT2_MEM_OFFSET (*(volatile char *const) (AMBA_BASE+7))
#define AMBA_SLOT1_AMBA_ADDR (*(volatile int *const) (AMBA_BASE+8))
#define AMBA_SLOT2_AMBA_ADDR (*(volatile int *const) (AMBA_BASE+12))

// 7-Segment Anzeige
#define SEG_BASE (0xFFFFFEE0)
#define DISPLAY (*(volatile char *const) (SEG_BASE+6))



#define SPI_BASE (0xF0000000)
#define SPI_CAPABILITY (*(volatile int *const) (SPI_BASE))
#define SPI_MODE (*(volatile int *const) (SPI_BASE+32))
#define SPI_MODE_test (*(volatile char *const) (SPI_BASE+34))
#define SPI_EVENT (*(volatile int *const) (SPI_BASE+36))
#define SPI_MASK (*(volatile int *const) (SPI_BASE+40))
#define SPI_COMMAND (*(volatile int *const) (SPI_BASE+44))
#define SPI_TRANSMIT (*(volatile int *const) (SPI_BASE+48))
#define SPI_RECEIVE (*(volatile int *const) (SPI_BASE+52))
#define SPI_SLAVE_SEL (*(volatile int *const) (SPI_BASE+56))

int main(int argc, char **argv)
{
  unsigned long i = 0x00000000;
  while (0==0){
    SPI_MODE = 0x07090000;
    SPI_MASK = 0x00004000;
    SPI_TRANSMIT = 0x12345678;
    DISPLAY = SPI_TRANSMIT;
    while((AMBA_CONFIG_C & 0x01) != 0x01){
      asm("nop");
```

```
        }
        i = SPI_RECEIVE;
        DISPLAY = SPI_RECEIVE;
        SPI_MODE = 0;
        SPI_EVENT = 0x00004000;
        SPI_MODE = 0x07090000;
        AMBA_CONFIG_C = 0;
        SPI_TRANSMIT = i;

        while((AMBA_CONFIG_C & 0x01) != 0x01){
          asm("nop");
        }
    }
}
```

Literatur

[1] M. Fletzer. *Diplomarbeit: Spear2 - An Improved Version of SPEAR*. TU Wien, Institut für Technische Informatik, 2008.(http://trac.ecs.tuwien.ac.at/Spear2/downloads)

[2] J. Mosser. *Diplomarbeit: AMBA4SPEAR2: An AMBA Extension Module for the SPEAR2 Processor Core*. TU Wien, Institut für Technische Informatik, 2008.(http://trac.ecs.tuwien.ac.at/Spear2/downloads)

[3] R.Seiger *miniUART Dokumentation*. TU Wien, Institut für Technische Informatik, 2005.