FH-OÖ Hagenberg/ESD
Hardware / Software Codesign, SS 2019
Florian Eibensteiner © 2019 (R 4744)

6. Übung: Virtual Prototyping – all about firmware

Name(n):

1 Interrupts und LT TLM

Wie in der letzten Übung bereits diskutiert, soll der VP nun um einen Interrupt erweitert werden. Interrupts sind asynchrone Unterbrechungen, die zu einem beliebigen Zeitpunkt auftreten können, den Programmfluss unterbrechen und eine Interrupt-Service-Routine (ISR) anstoßen. Da bei SystemC-Simulationen kooperatives Scheduling verwendet wird, kann die CPU allerdings nicht einfach unterbrochen werden. Daher muss entweder das CPU-Modell auf einer anderen Abstraktionsstufe implementiert werden, beispielsweise hinzufügen eines Instruction-Set-Simulator, oder man macht Abstriche bei der Verhaltensmodellierung eines Interrupts.

Erweitern Sie nun die CPU und das Target um einen Port vom Typ `bool`, um eine IRQ-Leitung zu modellieren. Das Target soll einen Interrupt auslösen sobald der Cordic-Core mit der Berechnung fertig ist und das Rdy-Flag gesetzt wird. Um in der CPU die ISR ausführen zu können, kann wie bei der Firmware ein Prozess implementiert werden, der via Funktionszeiger die Calback-Funktion aufruft. Überlegen Sie ob für das Interrupt-Handling im Target und der CPU `SC_METHOD` oder `SC_THREAD` verwenden. □

Der Funktionszeiger auf die ISR soll allerdings nicht im Konstruktor sondern zur Laufzeit über den Treiber des *XSCUGIC*s eingehängt werden. Hierzu kann die CPU um eine geeignete Set-Funktion erweitert werden. Sehen Sie auch eine geeignete Fehlerbehandlung für den Fall dass keine ISR eingehängt wurde vor.

2 Software

Wenn bei der HLS ein AXI4-Lite Slave-Interface hinzugefügt wird, werden automatisch einige C-Treiber-Files generiert. Eine Auflistung und Beschreibung welche Funktionen und Dateien generiert werden finden Sie in [Xil18, S.103ff, S.469ff]. Damit der Treiber später möglichst einfach auf die Zielplattform portiert werden kann, müssen am VP diese APIs ebenfalls zur Verfügung stehen.

Implementieren Sie die Module *xcordic* und *xcordic_sinit* [Xil18, S.104ff], wobei in unseren Fall die Funktionen `XCordic_LookupConfig`, `XCordic_Initialize`, `XCordic_CfgInitialize`, `XCordic_InterruptClear` und `XCordic_InterruptEnable` ausreichen – die letzten beiden Funktion können leer implementiert werden. Das Modul *xcordic_sinit* könnte wie folgt aussehen:

```
XCordic_Config *XCordic_LookupConfig(u16 DeviceId) {
  XCordic_Config *ConfigPtr = NULL;

  int Index;

  for (Index = 0; Index < XPAR_XCORDIC_NUM_INSTANCES; Index++) {
    if (XCordic_ConfigTable[Index].DeviceId == DeviceId) {
      ConfigPtr = &XCordic_ConfigTable[Index];
      break;
    }
  }

  return ConfigPtr;
}

int XCordic_Initialize(XCordic *InstancePtr, u16 DeviceId) {
  XCordic_Config *ConfigPtr;

  ConfigPtr = XCordic_LookupConfig(DeviceId);
  if (ConfigPtr == NULL) {
    InstancePtr->IsReady = 0;
    return (XST_DEVICE_NOT_FOUND);
  }

  return XCordic_CfgInitialize(InstancePtr, ConfigPtr);
}
```

Für die Verwendung des Interrupts muss die API des *xscugic* ebenfalls implementiert werden – orientieren Sie sich hierfür am BSP aus Übung 4. Definition der *Device_ID*, Basisadressen und der gleichen sollen in `xparameter.h` gemacht werden.

2.0.1 Treiber

Erweitern Sie den Treiber um eine Funktion für die Initialisierung des Cordic-Core. Definieren Sie eigenständig den Typ des Rückgabewertes *CordicStatus_t* und geben Sie einen entsprechenden Status zurück (z.B.: Fehler in der Initialisierung, ungültige Eingabewinkel). Verwenden Sie dafür beispielsweise folgenden Programmcode:

```
 // init cordic
CordicStatus_t cordic_init(){

  // call XCordic_Initialize here
  ...
  // setup interrupt handling here
}
```

Erweitern Sie nun die Schnittstelle für die Funktion zur Berechnung der Winkelfunktionen aus der letzten Übung ebenfalls um die Rückgabe eines Status:

```
CordicStatus_t CordicCalcXY(float const * const phi, float * const cos,
                            float * const sin, unsigned int * adr);
```

Zuletzt integrieren integrieren Sie das Interrupt-Handling in die Funktion, in dem Sie das Polling des Rdy-Flags durch ein Busy-Waiting ersetzen. Beachten Sie, dass an dieser Stelle in der Simulation ein Kontext-Switch erfolgen muss damit der Interrupt ausgelöst werden kann. Eine mögliche Implementierung könnte wie folgt aussehen:

```
#ifdef EMUCPUINSTANCE // define this within compiler options
#include <systemc.h>
#endif
...
// wait for IRQ is serviced
while(!rdy_irq){
#ifdef EMUCPUINSTANCE
  wait(...);
#endif
}
...
```

Literatur

[Xil18] Xilinx. *Vivado Design Suite User Guide – High-Level Synthesis*, December 2018.