

EE201A - Homework 6

Design of an interface handshake protocol for the IP checksum accelerator

In the previous steps, we did memory analysis and exploration of the protocol stack, and designed a stand-alone hardware acceleration unit that can evaluate and insert IP-packet checksums. There is one more step to take in the design flow of the protocol stack: integrating our hardware IP checksum evaluation unit back into the rest of the system, which still runs in software.

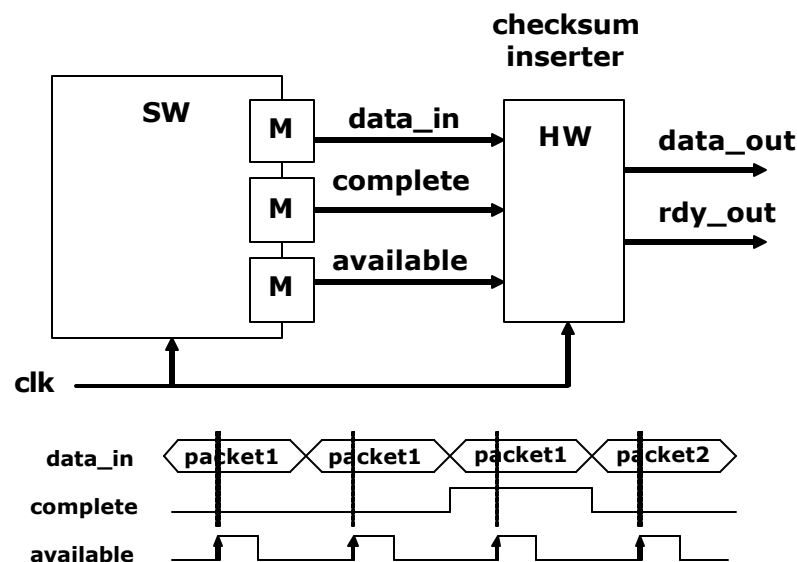


FIGURE 1. Handshake protocol to connect the checksum inserter hardware to a core running the rest of the TCP/IP protocol.

Figure 1 shows the integration problem that we are considering at this moment. The *checksum inserter* is a block for which we designed a dedicated FSM hardware model. The block processes packets which are provided out of an optimized software implementation of the HTTP/TCP/IP protocol stack. The interface between hardware and software is memory-mapped. This means that there are predefined addresses in the software memory space that map to the inputs of the checksum inserter. We also assume here that the core and the checksum inserter block run off a single synchronous clock.

Originally, the checksum inserter block had two input signals: *data_in* and *complete*. Each ‘clock cycle’, the block accepted one data byte from the input *data_in*, with *complete* indicating the last byte of a packet. At this moment it is clear that the rate of one byte-per-clock-cycle is not realistic when this stream is provided by software. Even when the bytes for *data_in* would be programmed out of a tight for-loop in software, each byte still takes several clock cycles to be provided since it is retrieved out of background memory (address calculations and access latencies). Therefore we introduce an extra signal between the core running the software and the checksum inserter block: *available*. This

signal will change from 0 to 1 whenever the core updates the value of *data_in* and *complete*. After *available* is set to 1, the checksum inserter block will wait for this signal to return to 0, before accepting the next data byte and flag from *data_in* and *complete*. Do not confuse *available* with a clock signal - it is simply a flag which is provided at the pace of the real clock *clk*. We call this kind of protocol a *single-sided handshake*. The checksum inserter block, being much faster than software, will always be able to accept the next data byte (We assume that the checksum inserter outputs will never block). Consequently, only a single synchronization signal is needed to keep it running in the same pace as the software. The effect of a single-sided handshake on the IP checksum controller is shown in Fig. 2. Extra transitions and states are introduced so that the handshake sequence is followed appropriately.

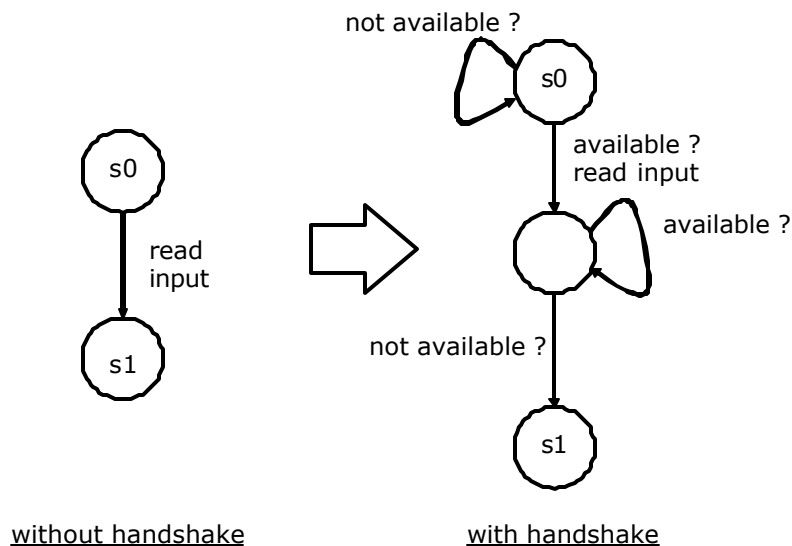


FIGURE 2. Effect of handshake on a read operation in the IP checksum controller

The way you will proceed in the design of the single-sided handshake depends on your target platform.

1. LEON

a) On the project homepage, you will find a package `hw6.zip` that contains a subdirectory `leon`. The directory contains the following files: `chkdrv.c` is the reference testbench in C, the program that runs on the block 'SW' in figure 1. `checksum.fdl` is a reference implementation of the checksum inserter block, as we have developed in HW5. This block does not contain the single-sided handshake protocol yet, but it does show how the hardware/software interface should be added. `makefile` shows how to compile the `chkdrv.c` file (which will run on top the the LEON instruction set simulator).

b) The goal is to modify `checksum.fdl` such that it interfaces to `chkdrv.c`. The software-side of the protocol is already defined. You only need to modify `checksum.fdl`. To simulate the software together with the hardware block, we will make use of another simulator called `fdl_tsim`. You will receive the server name that contains the `fdl_tsim` program by email. To run the cosimulation, you use the command:

```
> fdl_tsim checksum.fdl chkdrv.exe
```

The simulator will always run until the execution of the C program completes. Before you start modifying *checksum.fdl*, take a step back and think about the task ahead. The key the solution is to find a systematic way of transforming the original checksum FSM to one that contains a single-sided handshake protocol. This transformation is not hard if you do it systematically. If you make edits at random, you will get lost in complexity.

2. ARM

a) On the project homepage, you will find a package hw6.zip that contains a subdirectory arm. The directory contains the following files: *chkdrv.c* is the reference testbench in C, the program that runs on the block 'SW' in figure 1. *checksum.fdl* is a reference implementation of the checksum inserter block, as we have developed in HW5. This block does not contain the single-sided handshake protocol yet, but it does show how the hardware/software interface should be added. *makefile* shows how to compile the *chkdrv.c* file (which will run on top the the ARM instruction set simulator). The *sim* subdirectory contains the GEZEL-arm cosimulator. It will only run on Linux, for which you should have installed a cross-compiler earlier (HW1).

b) The goal is to modify *checksum.fdl* such that it interfaces to *chkdrv.c*. The software-side of the protocol is already defined. You only need to modify *checksum.fdl*. To run the cosimulation, you can use the command:

```
> make run
```

The simulator will always run until the C program completes execution. Before you start modifying *checksum.fdl*, take a step back and think about the task ahead. The key the solution is to find a systematic way of transforming the original checksum FSM to one that contains a single-sided handshake protocol. This transformation is not hard if you do it systematically. If you make edits at random, you will get lost in complexity

3. SpecC (TI/BLACKFIN)

For the design of this handshake protocol, we will make abstraction of the target platform (TI or BlackFin), but rather use the design features of SpecC.

a) On the project homepage, you will find a package hw6.zip that contains a subdirectory specc. The directory contains the following files: *checksum.sc* is the reference implementation of the checksum insertion processor in SpecC, as we have developed in HW5. It is given as a starting point to integrate the new protocol. *checksys.sc* is the reference testbench, that represents Fig 1. This program makes uses of a channel and interfaces to simulate the HW/SW connection. This file also contains an empty 'checksum' box, that should be filled out with a protocol-enabled version of *checksum.sc*.

b) The goal is to modify *checksys.sc* such that it includes the checksum verification processor. Before you start modifying *checksys.sc*, take a step back and think about the task ahead. The key the solution is to find a systematic way of transforming the original checksum FSM to one that contains a single-sided handshake protocol. This transformation is not hard if you do it systematically. If you make edits at random, you will get lost in complexity.

What to turn in

- *> Deadline: Thursday May 22, noon at Marilyn's desk
- *> One sheet of paper with your name and a link to your project homepage
- *> One your project homepage, post the modified code: *checksum.fdl* for the case of LEON and ARM, and *checksys.sc* for the case of SpecC.