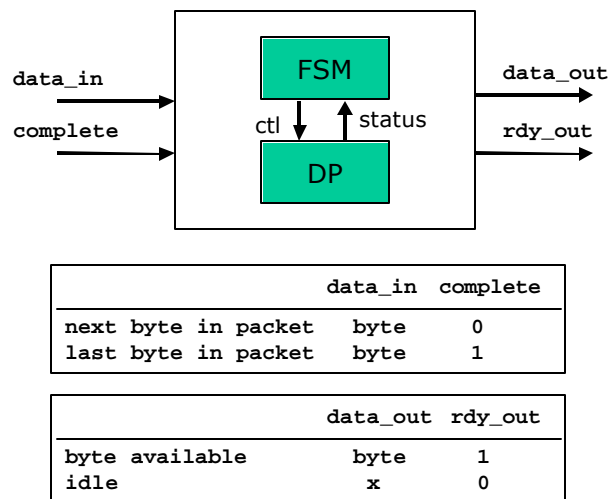


# EE201A - Homework 5

## Design of an accelerator for IP checksum evaluation

In Homework 4 we concluded that it is worthwhile to move system functionality around before implementing it. By performing IP checksum verification and insertion in dedicated offline processes, we were able to improve the memory access behavior of the TCP/IP protocol stack. *System Exploration* is always a good thing to do provided you define appropriate cost factors. In our case, the number of memory accesses to a central buffer was such a cost factor.



**Figure 1: Specification for the IP checksum insertion accelerator**

Now we are ready to consider further details of IP checksum evaluation. The C functions that designed for Homework 4 tell us *what* these blocks should do. In Homework 5, we will think about *how* to do it. Thus, we will move to modeling at a lower abstraction level. While there are different target platforms in use, we are not rushing to implementation yet. Instead, we will pick a modeling semantics - a stepping stone between high-level specification and implementation. The modeling semantics that we will use is FSMD (Finite State Machine Datapaths), a cycle-true modeling technique with split representation of data and control. While FSMD models are biased towards hardware implementation, they are still at a higher abstraction level than plain HDL models. As we will see later, the use of FSMD also eases integration of our block back into a system. To understand the need for an FSMD model, it is useful to consider the design trajectory that we are following. In the previous phases of the project, all design tasks were at high level of abstraction (in pure C code), and mostly of ‘top-down’. That is, we had virtually no constraints other than behavioral ones (like designing test vectors and minimizing memory accesses). In Homework 5, we step down to a lower level of abstraction for part of the system. This part will later have to be integrated back into the system, and result in a ‘bottom-up’ design task. It

is this integration aspect, and the need for cosimulation, that requires us to take a manageable modeling level.

The specification of the block we will develop is shown in Figure 1. IP packets will be provided over a byte-wide input `data_in`. Each clock cycle, a new data byte will be available. This is just an assumption that we make for now to make the development easier. (Later on, we will also consider interfacing and handshaking issues). The input `complete` is asserted (1) during the last data byte of a packet. The very first data byte to enter the block after startup can be considered to be the start of a packet. The processed data bytes are provided to an output `data_out`. Each cycle during which the `rdy_out` signal is asserted (1), the output `data_out` is valid. The function inside of the checksum insertion block is the same as developed for IP checksum insertion in `checksum.c` of Homework 4.

You will use one of the following FSM modeling mechanisms.

- \*> When your target platform is STRONGARM or LEON, you will use GEZEL.
- \*> When your target platform is BLACKFIN or TI, you will use SPECC.

To complete the homework, take the following steps.

1. Go to the EE201A Project Homepage and download the package `uip_hw5` to your SEASNET account. You will find two subdirectories - GEZEL and SPECC.
2. Both GEZEL and SPECC are installed in SEASNET. The makefiles in each of the GEZEL and SPECC subdirectories show how to run the simulators. Before you run SPECC, make sure you source the `settings.csh` file:  

```
source /w/class.01/ee/ee201a/cbin/specc/bin/setup.csh
```
3. Both of the GEZEL and SPECC subdirectories contain a sample testbench with an 'empty' checksum insertion block. The sample testbench is a 'PING' packet and an 'HTTP GET' packet, which are fed in alternately to the checksum insertion block. The testbench prints out the outputs of the checksum insertion block. You can directly edit into this testbench file (`checksum.fdl` for GEZEL and `checksum.sc` for SPECC).
4. Your task is to design an FSM that performs checksum insertion. Consult the project homepage for links on FSM modeling if you need more examples.

## What to turn in

- \*> Deadline: Tuesday May 13, 2003, 12 noon at Marilyn's desk.
- \*> On your project homepage, post the modified `checksum.c`.
- \*> Good Luck, use the Class Bulletin Board for any questions!