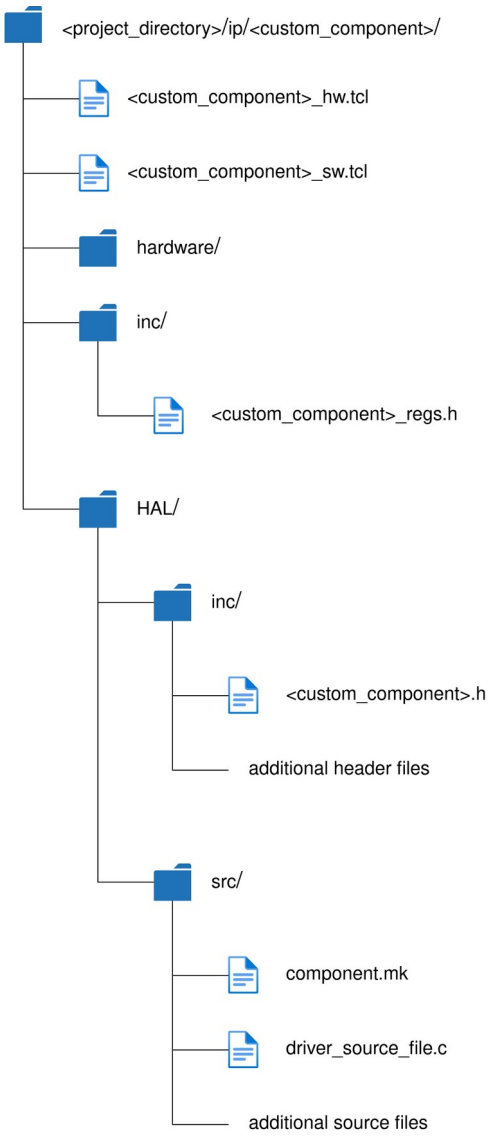
IP COMPONENTS

Every IP component is associated to a set of files characterized by the following directory structure:



The ***<component_name>_hw.tcl*** file describes the hardware side of the component. It can be created using the Platform Designer Component Editor and it includes the following elements:

- **Information** about the component, such as name, version and author
- **HDL description** of the component's hardware
- Constraint files (***sdc*** and/or ***qip***) that define the component for synthesis and simulation
- Component's interfaces, including I/O signals
- Parameters that configure the operation of the component

Using the Component Editor we can enter information, parameters, constraint files, signals and interfaces. Then, it is possible to automatically generate the top-level HDL file (go to the *Files* tab and click on *Create Synthesis File from Signals*), which must be edited to add the logic that describes the component's behavior,

and the whole *<component_name>_hw.tcl* file (clicking on *Finish*). All the HDL files and the constraint files are inserted in the *hardware/* directory.

While creating a custom memory IP with the Component Editor, it is important to set *isMemoryDevice* to 1 in the Avalon Interface (*Signals and Interfaces* tab → *Assignments*).

An IP component can be associated (although it is not strictly necessary) to a custom **HAL device driver** able to interface it to the HAL, so that it can be easily managed in SBT while creating the software for a Nios II system. Every HAL device driver has its own software description file, named ***<component_name>_sw.tcl***, which must be created manually (see *Nios 2 Software Developer's Handbook* for the list of commands) and a header file defining the hardware interface of the component, called ***<component name>_regs.h*** (which provides access to the interface registers by means of dedicated IORD/IOWR functions).

When a custom IP component is created using the Platform Designer Component Editor, only the hardware files are present. If we include it in a Platform Designer System and we create an SBT project, we do not have a HAL device driver, therefore we do not have a *<component name>_regs.h* file to exploit in order to interface the component. If we do not want to create it, we can use a generic driver for a generic memory-mapped device called ***io.h***, which is automatically included in any SBT project. In particular, it contains the following generic *#define* directives:

```
#define IORD(BASE, REGNUM)
#define IOWR(BASE, REGNUM, DATA)
```