***High-Speed Image Processing System Targeted for Electron Readout***

**LBNL IC GROUP DESIGN NOTE**



**Author**: Carl Grace

**Date**: January 12, 2016

**Revision**: 3.0

**Carl Grace**

Date: 11/06/09

Rev.: 1

**HIPSTER Control and Configuration Software**

|  |  |  |
| --- | --- | --- |
| **REVISION NUMBER** | **KEY CHANGES** | **ENGINEER** |
| 1.0 | Initial draft | CRG |
| 2.0 | Added SPI and I2C routines to control test chips on the PCB (DACs and clock gen) | CRG |
| 3.0 | Tutorial information added | CRG |

Table . HIPSTER Control and Configuration Software documentation revision history.

# Software Description

HIPSTER can be configured and controlled using a set of Python functions and scripts supplied by LBNL. These Python functions constitute the HIPSTER Control Software. Among other capabilities, these functions can be used to:

1. configure HIPSTER
2. configure the DACs and Clock generators on the eval board
3. execute commands such as ADC calibration initiation and JESD204B soft reset
4. verify the SPI
5. control the DACs on the evaluation board
6. test, read out, and manipulate the on-chip calibration memories
7. analyze ADC data

The configuration scripts are intended to be executed using Python 2.7. They are mostly compatible with Python 3 (with the exception of the print statements) but require some clean-up before they can be used with Python 3.

# Control Model

HIPSTER is controlled via a client-server model. A client sends commands over TCP/IP to a server operating either in a small computer or an FPGA. The client contains all knowledge about HIPSTER and the server formats commands and relays them to the appropriate devices. A client-server model allows the reuse of the same client code across multiple server implementations and simplifies the server. Initially, the server will be implemented in a Raspberry Pi. If desired, the slow controls can be re-implemented inside the control FPGA for production test, for instance.

The client is implemented in Python. This allows configuration and control tasks to be scripted while also allowing the option for controlling HIPSTER interactively using the Python Interpreter. Four chips on the HIPSTER evaluation board can be controlled using the Slow Control Software: 1. HIPSTER, 2. the two DAC8568 16-bit octal DACs, and 3. the Si5338 Clock Generator. Each chip can be identified using a unique device ID. The device IDs of the various chips on the HIPSTER evaluation board are summarized in Table 2.

|  |  |
| --- | --- |
| **Chip Name** | **Device ID** |
| HIPSTER | 0 |
| DAC8568 #1 | 1 |
| DAC8568 #2 | 2 |
| Si5338 Clock Generator | 3 |

Table . Device IDs for the different chips on the HIPSTER evaluation board.

## HIPSTER

HIPSTER is configured via scripts of Python functions.

## DAC5868

The DAC5868 octal-DAC is controlled via a SPI interface. High-level functions to control the DAC are included in the HIPSTER Control software. Low-level control is implemented as a driver in the server that operates in the Raspberry Pi.

## Si5338

The Si5338 Clock Generator is controlled via an I2C interface. High-level functions to control the Clock Generator are included in the Slow Control software. The configuration file for the Si5338 is generated by the ClockBuilder Desktop application available for download from Silicon Laboratories. Low-level control is implemented as a driver in the server that operates in the Raspberry Pi. The control script (go\_hipster.py) expects the Si5338 configuration file is in the same directory as it is and expects that its name is RegisterMap.txt

# Required Files

The Python files that comprise the HIPSTER Control Software are described in Table 3.

|  |  |  |
| --- | --- | --- |
| **File Name** | **Client or Server?** | **Description** |
| hipster\_spi.py | Client | Primitive functions and macros used to develop HIPSTER test scripts. |
| configure\_si5338.py | Client | Specialized functions used to configure the Si5338 Clock Generator and to load in configuration files generated by ClockBuilder Desktop. |
| hipster\_server.py | Server | Implements a TCP/IP server and sends commands to HIPSTER, the DAC8568 chips, and the Si5338. |
| hipster\_spi\_ops.py | Server | Functions used by the server to communicate with HIPSTER. |
| pll\_design.py | N/A | Python routine used to calculate PLL settings. |
| adc\_analysis.py | N/A | Routines used to calculate ADC performance (noise, linearity, etc) from raw data. |

Table . Python files used by the HIPSTER Slow Control Software.

# Getting Needed Source Files

The required source files are hosted on Github. If you have Git installed you can clone the repository using the following command:

git clone https://github.com/crgrace/hipster\_test

If you do not have git installed and you choose not to install it, you can download the code as a zip file. Doing so makes it more cumbersome to update the code when bug fixes or new features are available. The code can be downloaded directly from: <https://github.com/crgrace/hipster_test>

# Additional Files

In addition to Python source, the GitHub repository includes the HIPSTER register map, documentation of the ribbon cable used to connect the Raspberry Pi to the HIPSTER evaluation board, and the latest version of this document.

# Required Linux Kernel Modules

The Server that runs on the Raspberry Pi requires that kernel modules that support SPI and SMbus communications are available. There are many tutorials online that show how to configure them. If necessary, contact LBNL for assistance.

# Additional Considerations

The Raspberry Pi requires power via microUSB. The IP address is assigned by DHCP. Please contact LBNL for name / password assistance.

# Running Configuration Scripts

HIPSTER and its test board are configured using scripts that run on the Raspberry Pi. To operate HIPSTER, both Server and Client scripts need to be run.

## Server

To begin, run the Server as superuser on the Raspberry Pi in the appropriate directory (where the server.py and hipster\_spi\_ops.py files are located). This can be done as follows (where % is the Linux prompt on the Raspberry Pi and >>> is the Python Interpreter prompt):

%sudo python

>>> import hipster\_server

>>> hipster\_server.Server()

## Client

On a client Linux machine, run the appropriate configuration script. For example, to configure HIPSTER to send test data, run go\_hipster.py on the client (there is no need to run as superuser). Make sure that the constant SERVERNAME in the file hipster\_spi.py is correct for your Raspberry Pi (if running the client on the same Raspberry Pi, set SERVERNAME to “localhost”). As an example, on the client machine you would type the following in the directory where the python files are located:

%python go\_hipster.py

If you want to control HIPSTER interactively you will need to import the appropriate functions into your python environment. To do this, type the following at the Linux prompt:

%python

>>> import hipster\_spi as h

Now, you can access all the functions in hipster\_spi using the prefix h. So, for example, to write 0x000A to register 2 you would type:

>>> h.writeRegister(2,0x000A)

See below for an explanation of commonly used configuration functions. It is highly recommended to read the source for go\_hipster.py and look at some of the functions in spi\_hipster.py to see how the code is organized and see what functions could be useful and avoid duplication of effort.

# HIPSTER Configuration Registers

HIPSTER contains 52 configuration registers and three special-purpose registers.

|  |  |
| --- | --- |
| **Register Number** | **Description** |
| REG0 through REG51 | Configuration Registers. See HIPSTER Datasheet. |
| REG52 | Write Data Register (WDR) |
| REG53 | Command Register (CR) |
| REG54 | Mailbox Register (MBR) |

Table . HIPSTER SPI Register Map.

The Correction Logic Register File comprises 384 distinct 16-bit registers. Each of the 24 ADCs has two registers (for the two gain estimates) for each of their first eight stages.

# Key Configuration Functions

The key configuration functions are summarized in Table 5. Many higher-level functions are available to simplify testing and configuration tasks. See the source code docstrings for documentation of these functions.

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| writeRegister(register,data) | Writes data to HIPSTER SPI Register. |
| readRegister(register,data) | Reads data from HIPSTER SPI Register. |
| setBitInRegister(register,bit) | Sets individual bit in a HIPSTER SPI Register. |
| clearBitInRegister(register,bit) | Clears individual bit in a HIPSTER SPI Register. |

Table . Key HIPSTER Configuration Functions.

# Function Usage

## writeRegister(register,data,deviceID=0)

writeRegister is used to write to one of HIPSTER’s 52 configuration registers, or to one of HIPSTERs special-purpose registers, or to the evaluation DACs. The default deviceID is 0, which corresponds to HIPSTER.

### Examples

writeRegister(1,1000) writes the decimal value 1000 to SPI REG1.

writeRegister(20,0x2a12) writes the hexidecimal value 2A12 to SPI REG20.

## readRegister(register,data)

readRegister is used to read one of HIPSTER’s 52 configuration registers or one of its special-purpose registers.

### Example

readRegister(12) returns the decimal value current loaded in SPI REG12.

## setBitInRegister(register,bit)

setBitInRegister is used to set to 1 a specific bit of one of HIPSTER’s 52 configuration registers.

### Example

setBitInRegister(12,0) sets bit 0 of SPI REG12 to 1.

## clearBitInRegister(register,bit)

setBitInRegister is used to clear to 0 a specific bit of one of HIPSTER’s 52 configuration registers.

### Example

clearBitInRegister(2,12) clears bit 12 of SPI REG2 to 0.