

---

# basil Documentation

*Release 3.2.0*

T

Jun 25, 2021



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Features . . . . .	3
1.2	Installation . . . . .	3
1.3	Simulation . . . . .	4
1.4	License . . . . .	4
<b>2</b>	<b>Hardware</b>	<b>5</b>
2.1	MIO (Multi IO Card) . . . . .	5
2.2	GPAC (General Purpose Analog Card) . . . . .	7
2.3	MIO3 (Multi IO Card USB3) . . . . .	7
2.4	LX9 . . . . .	8
<b>3</b>	<b>Firmware</b>	<b>9</b>
3.1	basil bus . . . . .	10
<b>4</b>	<b>Software</b>	<b>11</b>
4.1	Yaml configuration file . . . . .	11
4.2	Transfer Layer (TL) . . . . .	11
4.3	Hardware Layer (HL) . . . . .	12
4.4	Register Layer (RL) . . . . .	12
<b>5</b>	<b>Modules</b>	<b>13</b>
5.1	Driver . . . . .	13
5.2	Driver . . . . .	13
5.3	Driver . . . . .	14
5.4	Driver . . . . .	14
5.5	Driver . . . . .	14
5.6	Driver . . . . .	15
5.7	Driver . . . . .	15
5.8	Driver . . . . .	15
5.9	Driver . . . . .	15
5.10	Driver . . . . .	15
5.11	Driver . . . . .	15
5.12	Driver . . . . .	16
5.13	Driver . . . . .	16
<b>6</b>	<b>Examples</b>	<b>19</b>
6.1	spi . . . . .	19
6.2	gpio . . . . .	21
<b>7</b>	<b>Indices and tables</b>	<b>23</b>

<b>Python Module Index</b>	<b>25</b>
<b>Index</b>	<b>27</b>

Basil is a modular readout framework intended to allow simple and fast data acquisition systems (DAQ) design. It consists of different hardware components, FPGA firmware modulus and a Python based control software.

Contents:



## INTRODUCTION

Basil is a modular readout framework intended to allow simple and fast data acquisition systems (DAQ) design. It consists of different hardware components, FPGA firmware modulus and a Python based control software.

### 1.1 Features

#### Firmware:

- very simple single master [bus definition](#)
- multiple [basic modules](#) (ex. SPI, SEQ)
- multiple [interfaces](#) (UART, USB2, USB3, Ethernet)

#### Software:

- layer structure following hardware
- generation based on yaml file
- register abstract layer (RAL)
- simulator interface allows software test against simulated RTL (thanks to [cocotb](#) )

### 1.2 Installation

From host folder run:

```
python setup.py install
```

or

```
pip install -e "git+https://github.com/SiLab-Bonn/basil.git#egg=basil&subdirectory=host"
```

## 1.3 Simulation

Thank to [Chris Higgs](#) basil has a simulation interface (SiSim) with allow communication with simulator as if talking to real hardware.

**To make simulation one need:**

- verilog simulator (ex. [Icarus](#) )
- [cocotb](#) library
- set interface type to SiSim

Basil unit tests make extensive use of this feature. See tests folder.

## 1.4 License

If not stated otherwise.

**Host Software:** The host software is distributed under the BSD 3-Clause (“BSD New” or “BSD Simplified”) License.

**FPGA Firmware:** The FPGA software is distributed under the GNU Lesser General Public License, version 3.0 (LGPLv3).



## **HARDWARE**

Basil also allows easy integration with custom Hardware.

For our purpose we developed general purpose hardware components which connect to custom DUTs. The MultiIO board (MIO) implements the main FPGA for digital IO, memory resources, and the interface (USB 2.0) to the PC. To add analog functionality, the General Purpose Analog Card (GPAC) was developed. It connects to the MultiIO board and provides 4 programmable power supplies, 12 current sources, 4 voltage sources, 4 fast ADC channels, digital IO with scalable voltages, LVDS, and a programmable injection pulse generator. The device under test (DUT) will be connected with a custom PCB to the MIO/GPAC hardware.

- MultiIO - Digital IO card with FPGA and USB 2.0 interface
- MIO3 - Digital IO card with FPGA and USB 3.0 interface based on Enclustra [KX1](#) module
- GPAC - General Purpose Analog Card

### **2.1 MIO (Multi IO Card)**

The “S3 Multi IO System” is developed as an easy to use multi purpose digital IO card. It includes a free programmable Xilinx Spartan3 FPGA, SRAM Memory, USB2.0 Interface and a 8051 microcontroller with I2C and SPI functionality. It is designed to provide sufficient digital IO capability to any kind of daughter card.



**Features:**

**Silicon devices**

- Xilinx Spartan3 FPGA - XC3S1000 FG320 4C
- Cypress USB Controller - CY7C68013A 128AXC
- Cypress async. SRAM - CY7C1061AV33 10ZXC
- Programmable clock generator - Cypress CY22150

**IO connections**

- USB2.0 B-type as host interface
- Multi-IO-Connector with 80 user IO's (VccIO 1:2 V to 3:3 V)
- Agilent debug connector (1253-3620)
- JTAG connection
- RJ-45 connector for 2 LVDS transmitter and 2 LVDS receiver
- Header with I2C and SPI functionality
- Header with additional FPGA user IO's
- 3 buffered LVTTTL outputs with LEMO
- 3 buffered LVTTTL inputs with LEMO

**Power supply**

- via external 5V supply
- via USB cable

**Configuration capability**

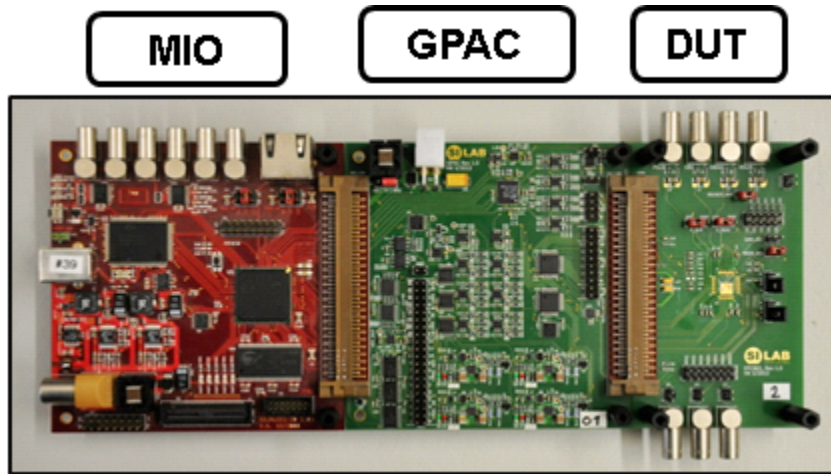
- via JTAG

- via USB2.0

[Documentation for MIO card.](#)

## 2.2 GPAC (General Purpose Analog Card)

GPAC Card is developed as an easy to use multi purpose analog IO card compatible with MIO Card.



### Features:

- 4 regulated power supplies, 0.8-1.83/2.83 V, max. 1000 mA, (controlled by I2C)
- 4 RX and 4 TX LVDS Lines
- 4 channel ADC, 25MS, 14bit
- 16 CMOS Outputs
- 8 CMOS Inouts
- 12 current source/sink, -1mA to +1mA, 12bit (controlled by I2C)
- 4 voltage outputs, 0-2.048 V, 12bit (controlled by I2C)
- 64x (4 available to DUT) channel slow ADC for monitoring (controlled by I2C)
- Injection Pulse Generator with programmable voltage levels (high and low)

[Documentation for GPAC card.](#)

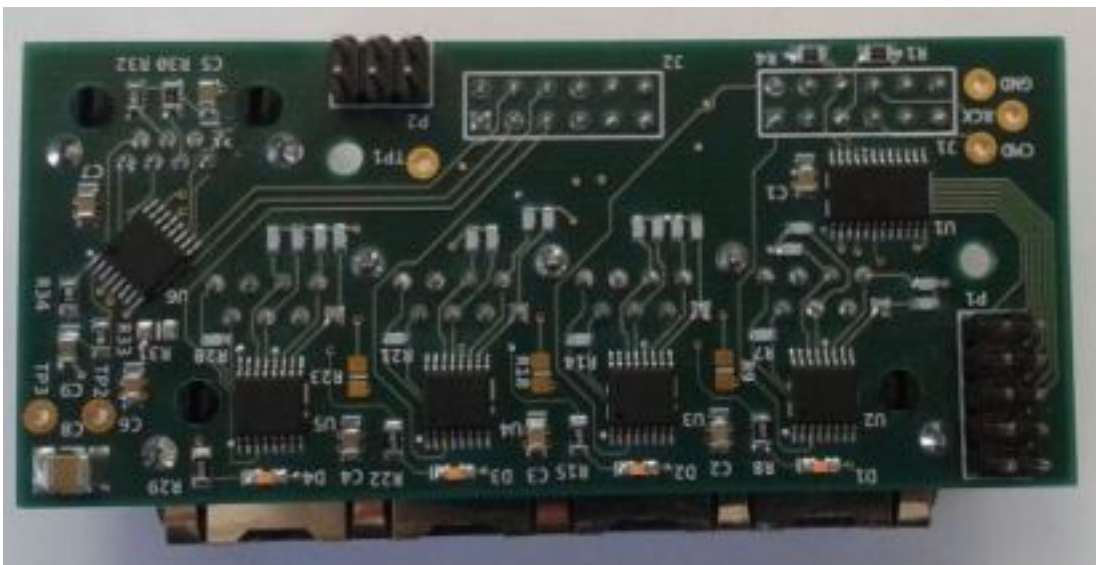
## 2.3 MIO3 (Multi IO Card USB3)

TBD.

## 2.4 LX9

LX9 Board.

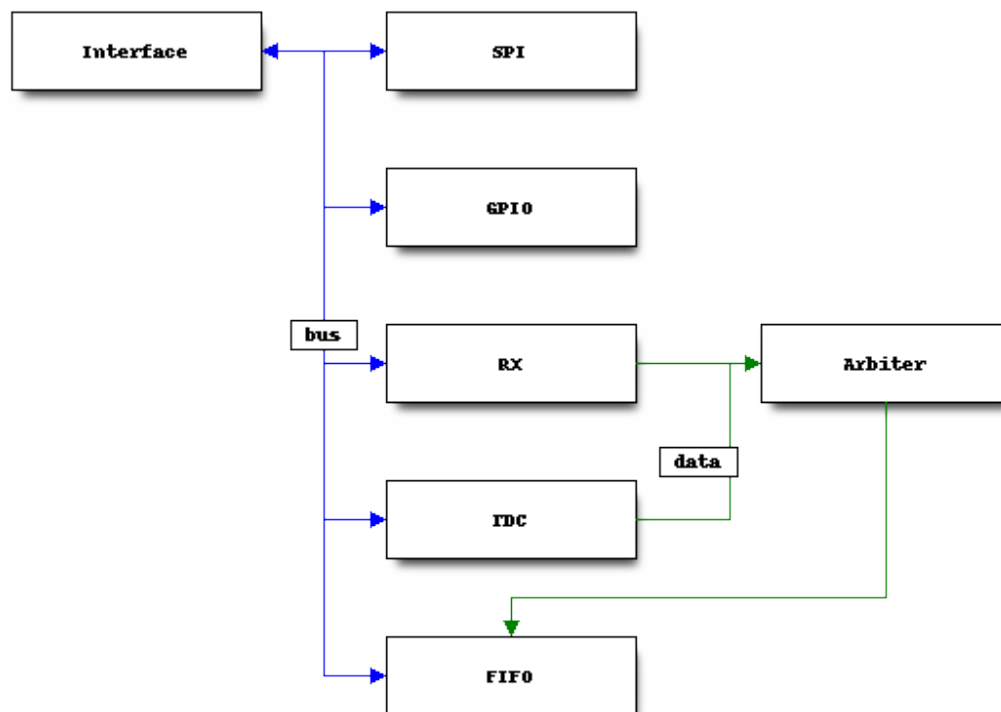
4 channel FE-I4 adepter with TLU:



## FIRMWARE

FPGA firmware consists of very simple single master bus definition and set of standard modules used by DAQ systems.

Typical firmware consists of basic bus connecting all modules. Control modules which provide configuration to DUT (like SPI/GPIO) and data taking modules (like data receivers). Received data (32 bit) are stored in the FIFO (large external memory) and can be continuously pulled from host application. Data from different modules are identified by source coding in 32bit data words.



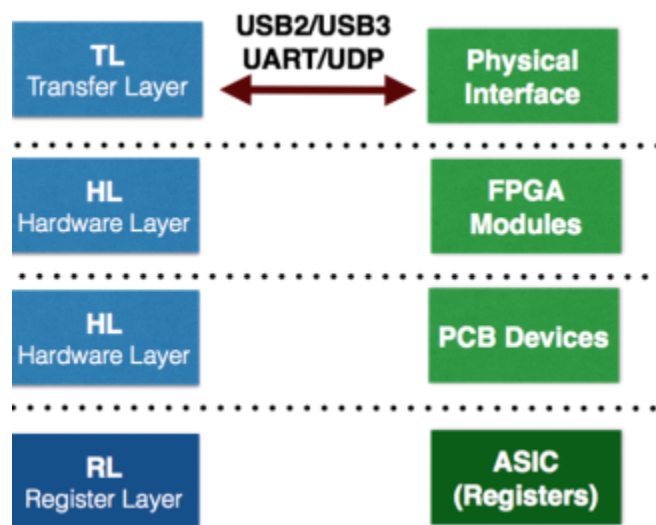
## 3.1 basil bus

single write

single read

## SOFTWARE

The software framework has a modular structure that reflects the firmware and adds extra layers to make hardware interface user friendly. It loosely follows Register Abstract Layer (RAL) concepts. All the layers are automatically created based on yaml configuration file.



### 4.1 Yaml configuration file

TBD

### 4.2 Transfer Layer (TL)

Implements communication interface like UART, USB, Ethernet or Simulation. Every TL interface implements 2 functions:

```
class basil.TL.TransferLayer.TransferLayer(conf)
    Transfer Layer implements minimum API needed access to hardware. On error raise IOError.

    init()
        Initialize and connect to hardware.

    read()
        Read access.
```

**Return type** None

**write**(*data*)

Write access.

**Parameters** **data** (*iterable*) – array/list of bytes

**Return type** None

## 4.3 Hardware Layer (HL)

Implements drivers for basil modules and external devices. Drivers can reference other drivers within HL.

## 4.4 Register Layer (RL)

Implements Register Level Abstraction. Allow to user/control software to work on DUT registers without taking thinking about underlying levels.



## MODULES

## 5.1 Driver

```
class basil.HL.gpio.gpio(intf, conf)
```

GPIO interface

```
reset()
```

Soft reset the module.

## 5.2 Driver

```
class basil.HL.spi.spi(intf, conf)
```

Implement serial programming interface (SPI) driver.

```
get_data(size=None, addr=None)
```

Gets data for incoming stream

```
get_en()
```

Gets state of enable.

```
get_repeat()
```

Gets Number of repetitions of sequence with delay 'wait' (if 0 -> repeat forever)

```
get_size()
```

Get size of shift register length

```
get_wait()
```

Gets time delay between repetitions in clock cycles

```
is_done()
```

Get the status of transfer/sequence.

```
reset()
```

Soft reset the module.

```
set_data(data, addr=0)
```

Sets data for outgoing stream

```
set_en(value)
```

Enable start on external EXT\_START signal (inside FPGA)

```
set_repeat(value)
```

If 0: Repeat sequence forever Other: Number of repetitions of sequence with delay 'wait'

**set\_size**(*value*)

Number of clock cycles for shifting in data ex. length of matrix shift register (number of pixels daisy chained)

**set\_wait**(*value*)

Sets time delay between repetitions in clock cycles

**start**()

Starts the shifting data

## 5.3 Driver

**class** basil.HL.seq\_gen.**seq\_gen**(*intf, conf*)

Sequencer generator controller interface for seq\_gen FPGA module.

## 5.4 Driver

**class** basil.HL.pulse\_gen.**pulse\_gen**(*intf, conf*)

Pulser generator

**get\_en**()

Return info if pulse starts with a fixed delay w.r.t. shift register finish signal (true) or if it only starts with .start() (false)

**set\_delay**(*value*)

Pulse delay w.r.t. shift register finish signal [in clock cycles(?)]

**set\_en**(*value*)

If true: The pulse comes with a fixed delay with respect to the external trigger (EXT\_START). If false: The pulse comes only at software start.

**set\_repeat**(*value*)

Pulse repetition in range of 0-255

**set\_width**(*value*)

Pulse width in terms of clock cycles

**start**()

Software start of pulse at random time

## 5.5 Driver

**class** basil.HL.seq\_rec.**seq\_rec**(*intf, conf*)

Sequencer receiver controller interface for seq\_rec FPGA module.

## 5.6 Driver

**class** basil.HL.cmd\_seq.**cmd\_seq**(*intf, conf*)  
 FEI4 Command Sequencer Controller Interface for cmd\_seq FPGA module.

## 5.7 Driver

**class** basil.HL.fei4\_rx.**fei4\_rx**(*intf, conf*)  
 FEI4 receiver controller interface for fei4\_rx FPGA module

## 5.8 Driver

**class** basil.HL.fast\_spi\_rx.**fast\_spi\_rx**(*intf, conf*)  
 Fast SPI interface  
**reset**()  
 Soft reset the module.

## 5.9 Driver

**class** basil.HL.tlu.**tlu**(*intf, conf*)  
 TLU controller interface

## 5.10 Driver

**class** basil.HL.tdc\_s3.**tdc\_s3**(*intf, conf*)  
 TDC controller interface

## 5.11 Driver

**class** basil.HL.sram\_fifo.**sram\_fifo**(*intf, conf*)  
 SRAM FIFO controller interface for sram\_fifo FPGA module.  
**property** **FIFO\_INT\_SIZE**  
 Get FIFO size in units of integers (32 bit).  
**fifo\_size** [int] FIFO size in units of integers (32 bit).  
**get\_FIFO\_INT\_SIZE**()  
 Get FIFO size in units of integers (32 bit).  
**fifo\_size** [int] FIFO size in units of integers (32 bit).  
**get\_data**()  
 Reading data in SRAM.  
**array** [numpy.ndarray] Array of unsigned integers (32 bit).

**get\_fifo\_int\_size()**  
*Deprecated* Get FIFO size in units of integers (32 bit).  
**fifo\_size** [int] FIFO size in units of integers (32 bit).

**get\_fifo\_size()**  
*Deprecated* Get FIFO size in units of bytes (8 bit).  
**fifo\_size** [int] FIFO size in units of bytes (8 bit).

**get\_read\_error\_counter()**  
*Deprecated* Get read error counter.  
**fifo\_size** [int] Read error counter (read attempts when SRAM is empty).

**get\_size()**  
*Deprecated*

## 5.12 Driver

**class** basil.HL.fadc\_rx.fadc\_rx(*intf, conf*)  
 Fast ADC channel receiver

**set\_align\_to\_sync**(*value*)  
 Align data taking to a synchronization signal, reset signal is the synchronization signal (hard coded connection in Verilog source code)

**set\_single\_data**(*value*)

## 5.13 Driver

**class** basil.HL.i2c.i2c(*intf, conf*)  
 Implement master i2c programming interface driver.

**property is\_ready**  
**raises** **ExceptionType** IOError  
 Transfer not acknowledged.

**read**(*addr, size*)  
 Read access.

**Parameters**

- **addr** (*char*) – i2c slave address
- **size** (*int*) – size of transfer

**Returns** data byte array

**Return type** array.array('B')

**write**(*addr, data*)  
 Write access.

**Parameters**

- **addr** (*char*) – i2c slave address
- **data** (*iterable*) – array/list of bytes

**Return type** None



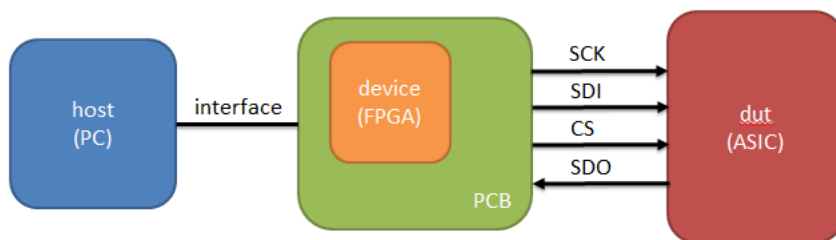
## EXAMPLES

Example of project can be found in [examples](#) folder.

For more usecases check also [tests](#) folder.

### 6.1 spi

An example shows how to create a simple spi interface.



Instantiate a verilog spi module in the firmware verilog code.

```
localparam SPI_BASEADDR = 32'h1000;
localparam SPI_HIGHADDR = 32'h1FFF;

spi
#(
    .BASEADDR(SPI_BASEADDR),
    .HIGHADDR(SPI_HIGHADDR),
    .MEM_BYTES(4)
) i_spi
(
    .BUS_CLK(BUS_CLK),
    .BUS_RST(BUS_RST),
    .BUS_ADD(BUS_ADD),
    .BUS_DATA(BUS_DATA),
    .BUS_RD(BUS_RD),
    .BUS_WR(BUS_WR),

    .SPI_CLK(SPI_CLK),

    .SCLK(SCLK),
    .SDI(SDI),
```

(continues on next page)

(continued from previous page)

```
.SDO(SDO),
.SEN(SEN),
.SLD(SLD)
);
```

Create a configuration file.

```
transfer_layer:
- name : intf
  type : SiSim

hw_drivers:
- name      : spi
  type      : spi
  interface : intf
  base_addr : 0x1000
  mem_bytes : 2

- name      : CNT
  type      : StdRegister
  hw_driver : spi
  size      : 16
  fields:
    - name : EN
      size  : 1
      offset : 15
    - name : OUT
      size  : 15
      offset : 14
```

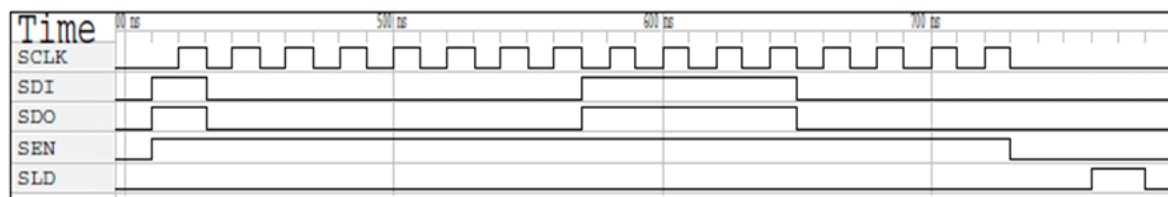
Write control program.

```
dut = Dut('spi.yaml')
dut.init()

dut['CNT']['EN'] = 1
dut['CNT']['OUT'] = 0x00f0
dut['CNT'].write()
dut['CNT'].start()

while not dut['CNT'].is_ready():
    pass
```

Result of simulation:



A workin example can be seen in tests/test\_SimSpi.py.



## 6.2 gpio

TBD.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### b

- `basil.HL.cmd_seq`, 15
- `basil.HL.fadc_rx`, 16
- `basil.HL.fast_spi_rx`, 15
- `basil.HL.fei4_rx`, 15
- `basil.HL.gpio`, 13
- `basil.HL.i2c`, 16
- `basil.HL.pulse_gen`, 14
- `basil.HL.seq_gen`, 14
- `basil.HL.seq_rec`, 14
- `basil.HL.spi`, 13
- `basil.HL.sram_fifo`, 15
- `basil.HL.tdc_s3`, 15
- `basil.HL.tlu`, 15
- `basil.TL.TransferLayer`, 11



## B

basil.HL.cmd\_seq  
     module, 15  
 basil.HL.fadc\_rx  
     module, 16  
 basil.HL.fast\_spi\_rx  
     module, 15  
 basil.HL.fei4\_rx  
     module, 15  
 basil.HL.gpio  
     module, 13  
 basil.HL.i2c  
     module, 16  
 basil.HL.pulse\_gen  
     module, 14  
 basil.HL.seq\_gen  
     module, 14  
 basil.HL.seq\_rec  
     module, 14  
 basil.HL.spi  
     module, 13  
 basil.HL.sram\_fifo  
     module, 15  
 basil.HL.tdc\_s3  
     module, 15  
 basil.HL.tlu  
     module, 15  
 basil.TL.TransferLayer  
     module, 11

## C

cmd\_seq (class in basil.HL.cmd\_seq), 15

## F

fadc\_rx (class in basil.HL.fadc\_rx), 16  
 fast\_spi\_rx (class in basil.HL.fast\_spi\_rx), 15  
 fei4\_rx (class in basil.HL.fei4\_rx), 15  
 FIFO\_INT\_SIZE (basil.HL.sram\_fifo.sram\_fifo property),  
     15

## G

get\_data() (basil.HL.spi.spi method), 13

get\_data() (basil.HL.sram\_fifo.sram\_fifo method), 15  
 get\_en() (basil.HL.pulse\_gen.pulse\_gen method), 14  
 get\_en() (basil.HL.spi.spi method), 13  
 get\_FIFO\_INT\_SIZE() (basil.HL.sram\_fifo.sram\_fifo  
     method), 15  
 get\_fifo\_int\_size() (basil.HL.sram\_fifo.sram\_fifo  
     method), 15  
 get\_fifo\_size() (basil.HL.sram\_fifo.sram\_fifo  
     method), 16  
 get\_read\_error\_counter()  
     (basil.HL.sram\_fifo.sram\_fifo method), 16  
 get\_repeat() (basil.HL.spi.spi method), 13  
 get\_size() (basil.HL.spi.spi method), 13  
 get\_size() (basil.HL.sram\_fifo.sram\_fifo method), 16  
 get\_wait() (basil.HL.spi.spi method), 13  
 gpio (class in basil.HL.gpio), 13

## I

i2c (class in basil.HL.i2c), 16  
 init() (basil.TL.TransferLayer.TransferLayer method),  
     11  
 is\_done() (basil.HL.spi.spi method), 13  
 is\_ready (basil.HL.i2c.i2c property), 16

## M

module  
     basil.HL.cmd\_seq, 15  
     basil.HL.fadc\_rx, 16  
     basil.HL.fast\_spi\_rx, 15  
     basil.HL.fei4\_rx, 15  
     basil.HL.gpio, 13  
     basil.HL.i2c, 16  
     basil.HL.pulse\_gen, 14  
     basil.HL.seq\_gen, 14  
     basil.HL.seq\_rec, 14  
     basil.HL.spi, 13  
     basil.HL.sram\_fifo, 15  
     basil.HL.tdc\_s3, 15  
     basil.HL.tlu, 15  
     basil.TL.TransferLayer, 11

## P

`pulse_gen` (class in `basil.HL.pulse_gen`), 14

## R

`read()` (`basil.HL.i2c.i2c` method), 16

`read()` (`basil.TL.TransferLayer.TransferLayer` method),  
11

`reset()` (`basil.HL.fast_spi_rx.fast_spi_rx` method), 15

`reset()` (`basil.HL.gpio.gpio` method), 13

`reset()` (`basil.HL.spi.spi` method), 13

## S

`seq_gen` (class in `basil.HL.seq_gen`), 14

`seq_rec` (class in `basil.HL.seq_rec`), 14

`set_align_to_sync()` (`basil.HL.fadc_rx.fadc_rx`  
method), 16

`set_data()` (`basil.HL.spi.spi` method), 13

`set_delay()` (`basil.HL.pulse_gen.pulse_gen` method),  
14

`set_en()` (`basil.HL.pulse_gen.pulse_gen` method), 14

`set_en()` (`basil.HL.spi.spi` method), 13

`set_repeat()` (`basil.HL.pulse_gen.pulse_gen` method),  
14

`set_repeat()` (`basil.HL.spi.spi` method), 13

`set_single_data()` (`basil.HL.fadc_rx.fadc_rx`  
method), 16

`set_size()` (`basil.HL.spi.spi` method), 13

`set_wait()` (`basil.HL.spi.spi` method), 14

`set_width()` (`basil.HL.pulse_gen.pulse_gen` method),  
14

`spi` (class in `basil.HL.spi`), 13

`sram_fifo` (class in `basil.HL.sram_fifo`), 15

`start()` (`basil.HL.pulse_gen.pulse_gen` method), 14

`start()` (`basil.HL.spi.spi` method), 14

## T

`tdc_s3` (class in `basil.HL.tdc_s3`), 15

`tlu` (class in `basil.HL.tlu`), 15

`TransferLayer` (class in `basil.TL.TransferLayer`), 11

## W

`write()` (`basil.HL.i2c.i2c` method), 16

`write()` (`basil.TL.TransferLayer.TransferLayer` method),  
12