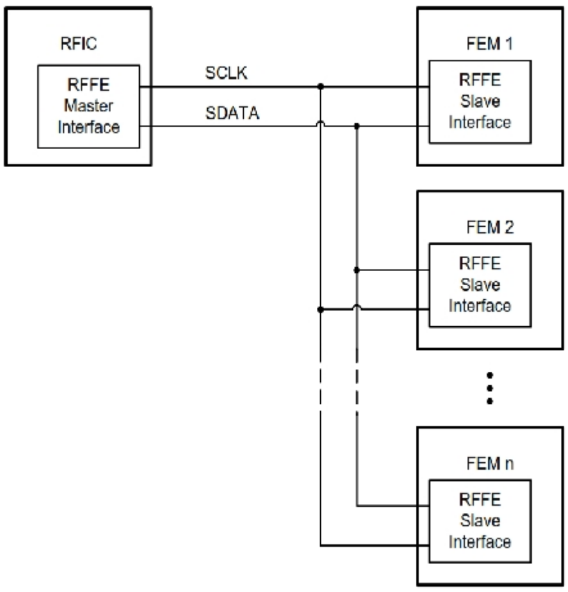
Using SPI Storm with a custom protocol

Many serial protocols used for chip-to-chip communications are very similar to the 'Serial Peripheral Interface' (SPI).

This paper shows how to use SPI Storm custom protocol engine to define and use a serial protocol that is different from SPI. The MIPI RF Front-End (RFFE) Control Interface is taken as an example

MIPI RF Front-End (RFFE) Control Interface

The 'MIPI' (Mobile Industry Processor Interface) Alliance (www.mipi.org) is an organization that develops interface specifications for the mobile industry. It was founded in 2003 by ARM, Intel, Nokia, Samsung, STMicroelectronics and Texas Instruments.



MIPI specifications provide interface solutions for mobile handsets. MIPI is divided into multiple working groups. The MIPI RF Front-End working group's scope is to define and standardize control interface solutions for RF front-end components and modules. This includes numerous front-end devices including power amplifiers, low-noise amplifiers, filters, switches, power management module, antenna tuners and sensors.

The RFFE Control Interface is a master-slave two wire interface composed of a single bidirectional data line and a clock signal generated by the interface master. It runs up to 26 MHz and can count up to 15 slave devices per bus master.

SPI vs. RFFE

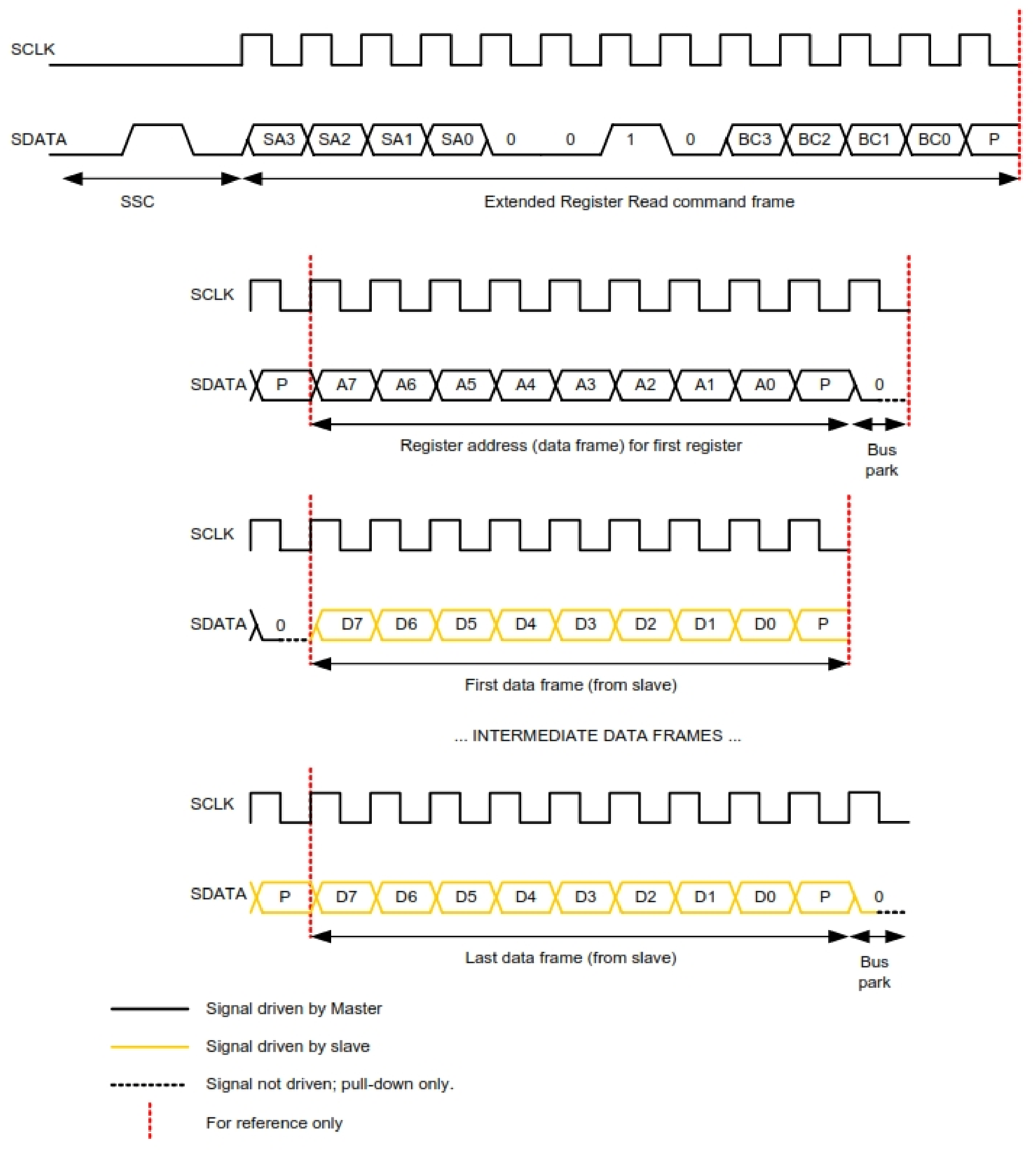
In its publications (example: http://mipi.org/sites/default/files/RFFE%20Marketing%20Overview_0.pdf) , the MIPI RFFE working group points out that the standard Serial Peripheral Interface (SPI) is not suitable for their use: SPI slave devices cannot share the bus lines. Without the proper slave addressing on a shared data bus, SPI requires one slave select line and one slave-to-master data line per slave.

SPI is therefore characterized by poor topology and bad system scaling - hence the need for another protocol.

The table below shows the main points of comparison between SPI and RFFE interface.

SPI	RFFE
> 4-wires interface (SCLK, MOSI, MISO, 1x SS per slave)	2-wires interface: SCLK, SDATA (bidirectional).
Slave select line	No slave select line - slave addressing scheme
Single master	Single master
Max. speed not defined	Up to 26 MHz
Max. nr of slaves not defined	Up to 15 slaves

Extended Register Read at Full Speed - an overview


SCLK
SDATA
SA3
SA2
SA1
SA0
0
0
1
0
BC3
BC2
BC1
BC0
P
SSC
Extended Register Read command frame
SCLK
SDATA
P
A7
A6
A5
A4
A3
A2
A1
A0
P
0
Register address (data frame) for first register
Bus park
SCLK
SDATA
0
D7
D6
D5
D4
D3
D2
D1
D0
P
First data frame (from slave)
... INTERMEDIATE DATA FRAMES ...
SCLK
SDATA
P
D7
D6
D5
D4
D3
D2
D1
D0
P
0
Last data frame (from slave)
Bus park
Signal driven by Master
Signal driven by slave
Signal not driven; pull-down only.
For reference only

RFFE 'Extended Register Read at full speed' command shown above is one of the most complete types of commands defined for the RFFE protocol. **Almost all the other RFFE commands are simpler versions of it.** This paper shows how to implement this command with SPI Storm.

On the figure, the yellow color marks when the slave drives the data line. It must be noted that the clock is always driven by the master. All the bits defined for this command are *synchronous* to the SCLK.

In this example, SPI Storm will be programmed to be used as the RFFE bus master.

SPI Storm custom protocol engine definition

SPI Storm is able to read / write with user-defined serial protocols. Each protocol can be defined as a sequence of **'segments'.** Each segment is defined according to the following properties:

- The segment length as a number of clock cycles;
- Whether the clock signal runs or is it held at a constant level during the segment execution;
- The data lines that are used for the segment;
- Each data line usage during the segment execution:
    - → Is the data line held at a constant level or floating?
    - → Is the data line used as an output of SPI Storm? If so, what is the value of the data generated during the segment execution?
    - → Is the data line used as an input of SPI Storm? If so, will the data line be sampled while the segment is executed?

Segments are really to be considered as the **'building blocks'** of any specific protocol. Segments can be grouped as 'macros'. When SPI Storm sends a read or write command to the defined serial interface, it executes a sequence of pre-defined macros. This enables segment reuse, as a the same segment can be used for several macros.

Hence, defining a custom protocol in with SPI Storm consists in 'cutting' the protocol into slices that represent segments with specific properties and assemble them as macros. There is are many alternatives to define any given protocol segments and macros.

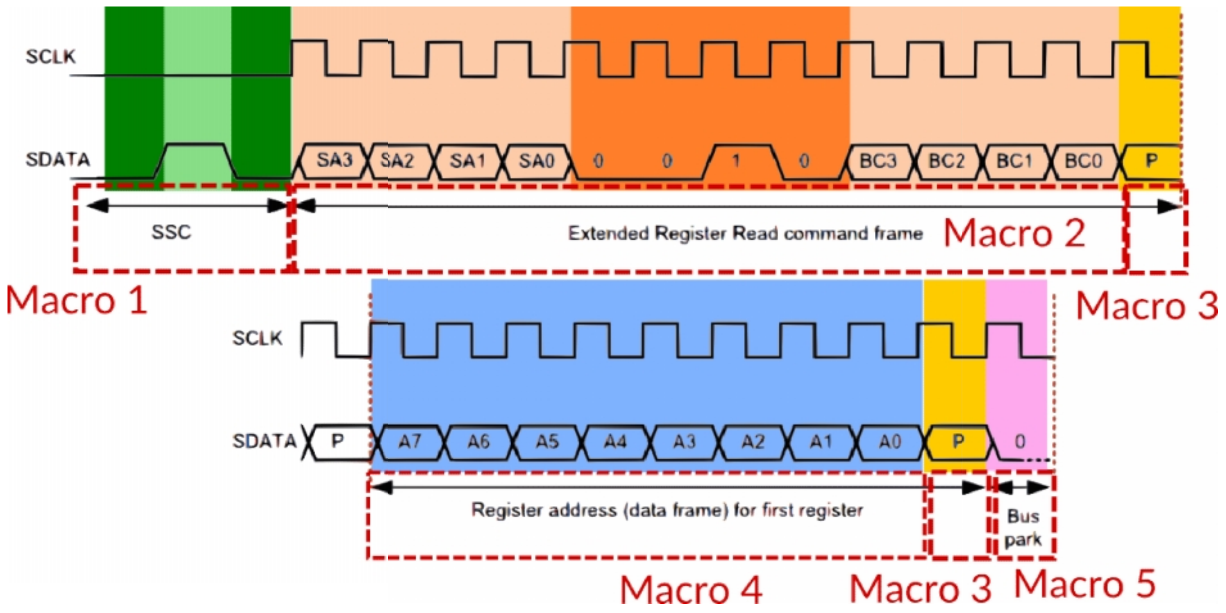
Macros can be based on convenient definitions that match a protocol specification, on data size that can be easily computed, and many user-defined criteria. SPI Storm and SPI Storm Studio provide lots of flexibility to describe the protocol. Once it is executed, SPI Storm studio 'compiles' the description so it fits into the SPI Storm USB device's memory and so its time of execution is optimized.

In the following sections, this paper provides an example for describing the 'Extended Register Read at full speed' command of the RFFE interface protocol.

RFFE read part 1:

Commands and addresses sent by the master to initiate the access.

This piece of the protocol is shown on the picture below:



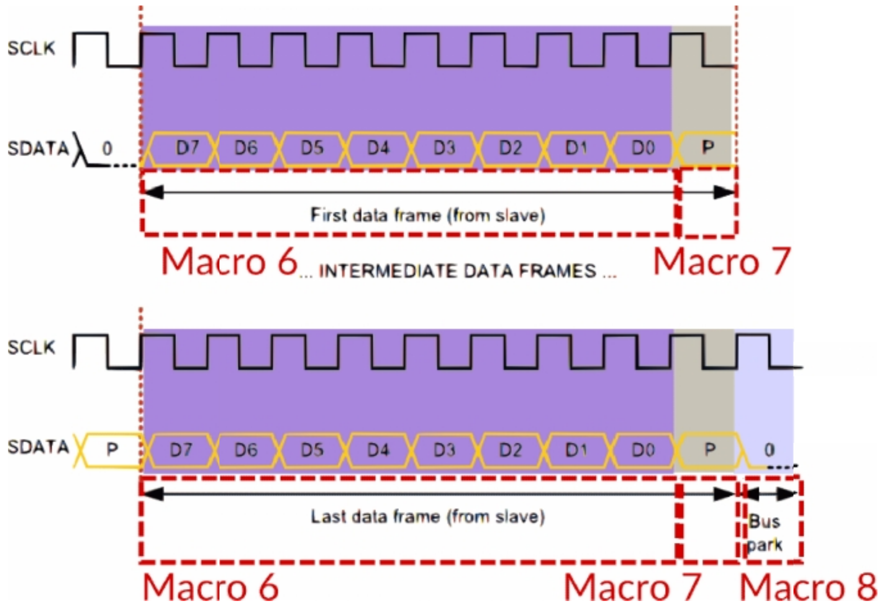
A macro is defined for each of the following parts of the protocol:

Macro	Macro name	Segments list	Segments names
Macro 1 'SSC'	SSC	1 bit logic '0' 1 bit logic '1' 1 bit logic '0'	Logic0NoClock Logic1NoClock Logic0NoClock
Macro 2 'Extended Register Read command frame'	ExtRegRead	4 bits variable slave address (SA3:SA0) 4 bit constant control value "0010" 4 bit variable byte count (BC3:BC0)	VarNibble VarNibble[1] VarNibble
Macro 3 Master Parity	MasterParity	1 bit parity	VarParity
Macro 4 Register Address	RegAddr	8 bits address	VarByte
Macro 5 Master Bus Park	MasterBusPark	1 bit 'Bus Park'	BusPark

TOTAL: 5 Macros based on 6 Segments.

[1] Even if the nibble always takes the value "0010", we use the same VarNibble segment here, to which we'll assign this value. An alternative way would be to assemble 4 separate segments of 1 bit with a constant value.

RFFE read part 2:
Reading back data from the slave



Macro	**Macro name**	**Segments list**	**Segments names**
Macro 6 'Data Frame'	DataFrame	8 bits Data from slave	SlaveDataByte
Macro 7 'Slave Parity'	SlaveParity	1 bit Slave parity	SlaveParity
Macro 8: 'Slave Bus Park'	SlaveBusPark	1 bit Slave Bus Park	SlaveBusPark

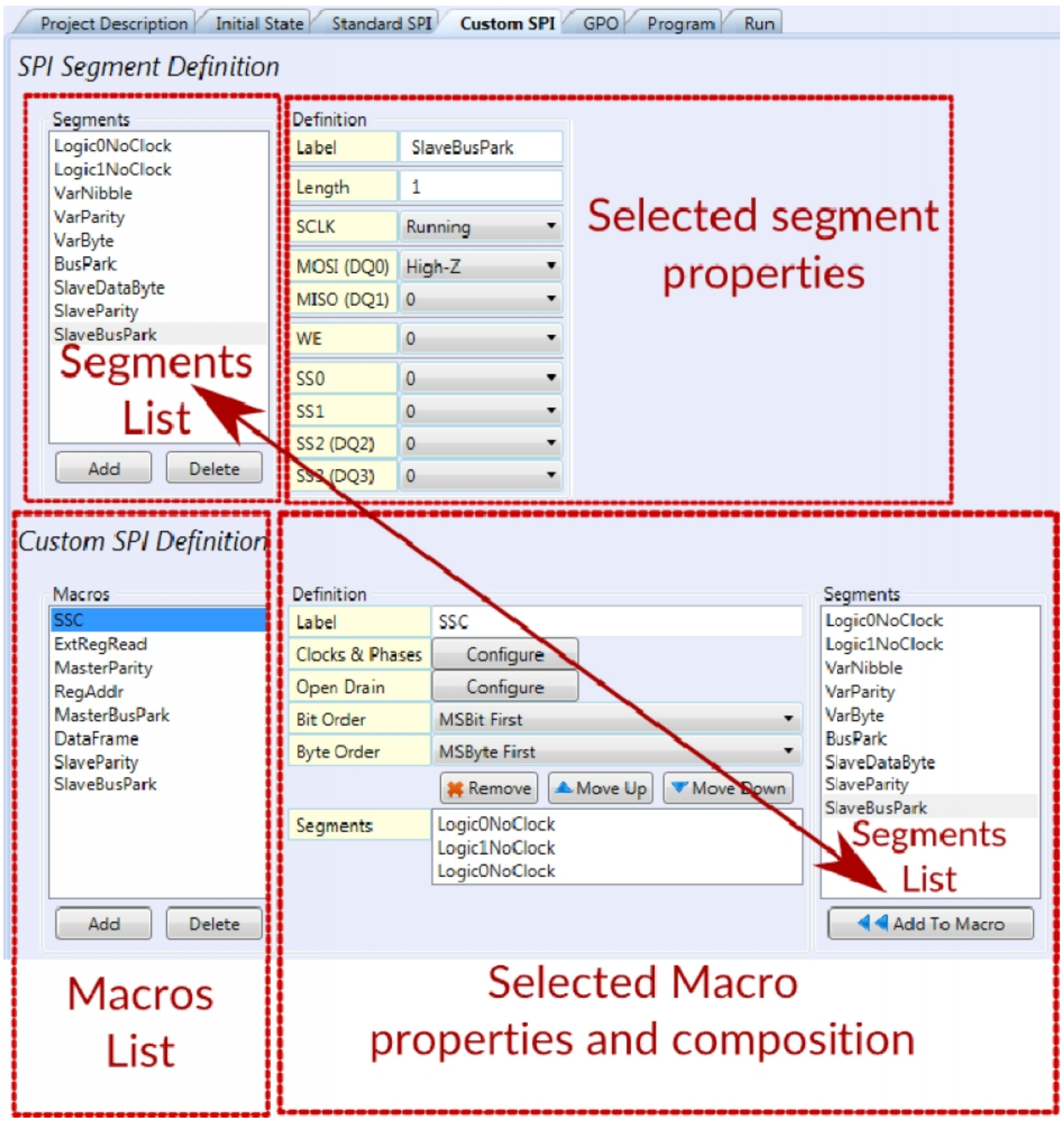
TOTAL: 3 Macros based on 3 Segments.

Defining the protocol with SPI Storm Studio

Please refer to SPI Storm Studio user's guide for details about how to use SPI Storm Studio (http://www.byteparadigm.com/files/documents/SPIStormStudio_UsersGuide.pdf)

SPI Storm Studio and the reference project (.ssp) file can be freely downloaded from the following link: http://www.byteparadigm.com/files/download/MIPI-RFFE.ssp.

The picture below shows a screen capture of SPI Storm Studio's 'Custom SPI' tab, from which the segments and macros can be defined. It shows the list of segments and macros and their properties.



Segments

LogicONoClock
Logic1NoClock
VarNibble
VarParity
VarByte
BusPark
SlaveDataByte
SlaveParity
SlaveBusPark

Definition

Label	SlaveBusPark
Length	1
SCLK	Running
MOSI (DQ0)	High-Z
MISO (DQ1)	0
WE	0
SS0	0
SS1	0
SS2 (DQ2)	0
SS3 (DQ3)	0

Macros

SSC
ExtRegRead
MasterParity
RegAddr
MasterBusPark
DataFrame
SlaveParity
SlaveBusPark

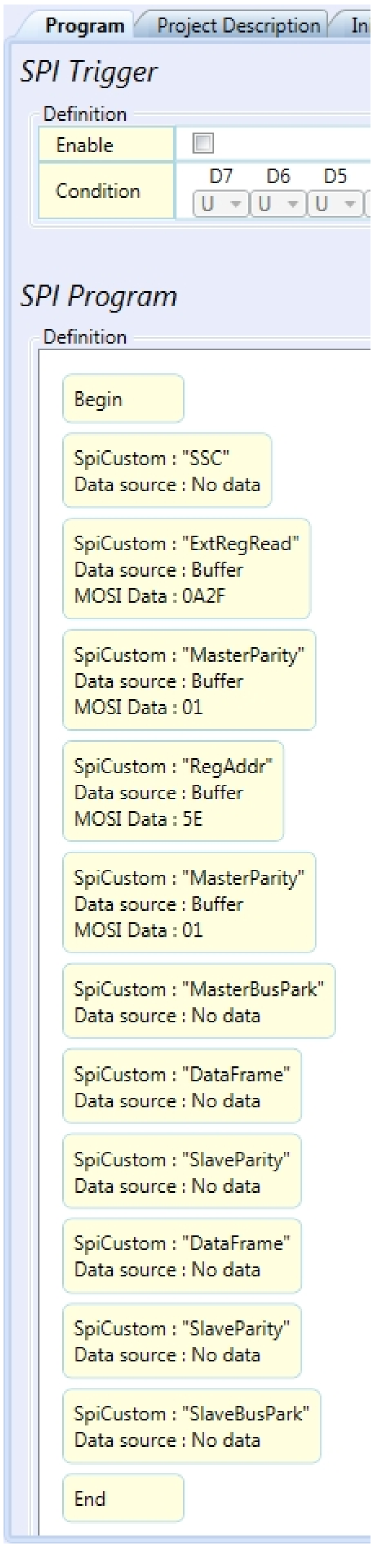
Definition

Label	SSC
Clocks & Phases	Configure
Open Drain	Configure
Bit Order	MSBit First
Byte Order	MSByte First
Segments	LogicONoClock Logic1NoClock LogicONoClock

Segments

LogicONoClock
Logic1NoClock
VarNibble
VarParity
VarByte
BusPark
SlaveDataByte
SlaveParity
SlaveBusPark

Sequencing the macros as a program with SPI Storm Studio



Program | Project Description | In

SPI Trigger

Definition

Enable ☐

Condition D7 U D6 U D5 U

SPI Program

Definition

Begin

SpiCustom : "SSC"
Data source : No data

SpiCustom : "ExtRegRead"
Data source : Buffer
MOSI Data : 0A2F

SpiCustom : "MasterParity"
Data source : Buffer
MOSI Data : 01

SpiCustom : "RegAddr"
Data source : Buffer
MOSI Data : 5E

SpiCustom : "MasterParity"
Data source : Buffer
MOSI Data : 01

SpiCustom : "MasterBusPark"
Data source : No data

SpiCustom : "DataFrame"
Data source : No data

SpiCustom : "SlaveParity"
Data source : No data

SpiCustom : "DataFrame"
Data source : No data

SpiCustom : "SlaveParity"
Data source : No data

SpiCustom : "SlaveBusPark"
Data source : No data

End

SPI Storm Studio's 'Program' tab is where a 'program' can be defined.
A program is a sequence of macros ready to be executed.

Simple mouse controls allows inserting and organizing the custom macros in the program tab area.

If the macro being inserted requires specifying a data, a window pops up to enter it. Variable data can be entered directly from SPI Storm Studio's interface or provided from files.

Please refer to SPI Storm Studio user's guide for more information about how to use it.
(http://www.byteparadigm.com/files/documents/SPIStormStudio_UsersGuide.pdf)

The screen capture on the left shows the list of macros that will be sequentially executed to send an **'Extended Register Read at Full Speed'** command from the RFFE interface protocol.

Worth noting, each macro can also be called independently from the API provided with SPI Storm Studio. Please refer the 'SPI Storm Studio C library user's guide' for more information.
(http://www.byteparadigm.com/files/documents/ug_SPIStormStudio_CLib.pdf)

Once a program is complete, it can be run from **SPI Storm Studio's Run Tab.**

About SPI Storm

Byte Paradigm's SPI Storm is one of the most advanced USB serial protocol host adapter available today.
It is used for functional testing on systems using SPI and many other serial protocols.

Unlike its competitors, **SPI Storm goes up to 100 MHz** and **includes a custom protocol definition engine** that provides **full control** of when you want to sample data from any SPI(-like) slave. Hence, it can easily handle the situations described in this paper.

For more information about SPI Storm, please go to: http://www.byteparadigm.com/products/spi-storm/