Tester protocol description

The communication between the computer and the Tester can be described as Master/Slave. This means that the Tester will never send any data without a request from the computer.
For clarity we define the following terms.

- *Core Board*, the board with the 6809 µP, RAM, EPROM, RS-232 interface, etc.
- *I/O Board*, the board that connects to the Core Board
- *Tester Board*, the board that is piggy-back installed on the I/O Board
- *Connector Board*, the board with the connector for the FlipChip to be tested
- *Tester*, the combination of one Core Board, one or two I/O Board(s) / Tester Board(s) and one Connector Board
- *Module Under Test*, (MUT), the FlipChip seated in the connector on the Connector Board
- *input* and *output*, defined with the FlipChip as point of view
  input for the FlipChip means that the Tester is an output for that contact finger, and vice versa

The Tester recognizes the following commands.

- Define contact finger I/O direction
- Put data pattern on (input) contact fingers of the MUT.
- Query data pattern on (input), send result of (output) all contact fingers of the MUT.
- Set Tester in Safe State
- Enable Tester hardware
- Report Tester State

The first 3 commands, issued by the computer, start with an alphabetic letter and are followed by 4 bytes. These 4 bytes represent the 32 bits of data that match the available 32 contact fingers on both sides of the MUT. See the “Byte / Contact Finger” table. (Remark. A FlipChip can have 18 contact fingers on each side of the circuit board, but the I/O Board and Tester Board hardware is limited to 32 I/O channels. This poses no real limitations to the Tester, because 4 contact fingers have a fixed definition, see the “Byte / Contact Finger” table.)
As a FlipChip to be tested can have a single-height or double-height form factor, the number of contact fingers can be either 32 or 64. The defined commands only have 4 bytes of data, and to identify for which socket of the connector on the Connector Board the data is intended, the command letter is either a uppercase or lowercase alphabetic.
In the command description, all commands are displayed for socket “A” (with the uppercase letter). The same command is valid for socket “B” when the lowercase letter is used.
Note that the “Set” and the “Enable” command apply to the Tester, so there is no lowercase letter for these two commands.
The Tester will always send a reply to the computer after the Tester receives any command. The information of the reply will always contain a single byte checksum, but the reply can also contain data bytes. This checksum is a simple 8-bit addition without overflow, and then 2’s-complemented, of the received and sent data. The computer verifies that no error occurred in the communication by the 8-bit addition without overflow of all sent data bytes and received data bytes (including the checksum). The result of the addition is zero, if during the communication no errors occurred. See the paragraph “Checksum calculation” for a detailed description of the checksum mechanism.

Define contact finger I/O direction

The “D”, ‘Define I/O direction’ command defines which contact fingers, of the FlipChip to be tested, are inputs and which are outputs. The command letter is followed by 4 bytes that actually define each contact finger. The 4 bytes (32 bits) align with the contact fingers as described in the “Byte / Contact Finger” table. A ‘1’-bit indicates that the corresponding contact finger is an input, a ‘0’-bit indicates that the corresponding contact finger is not a FlipChip input. FlipChip inputs will be driven with values by the Tester “P” and “Q” commands described below.
When the Tester receives the “D” command, the Tester responds with a single byte. This byte, called the checksum, is the acknowledgement for the computer that the “D” command is accepted. The byte is

the 2's-complement of the addition without overflow of the received command letter and the 4 bytes of data, and as such the computer can check that the command is received correctly. In the following representation "wxyz" represent the 4 bytes of data, "c" the checksum byte.

Computer	Flow	Tester
send command "Dwxyz" await response receive checksum and verify	→ ← 	receive 'Define I/O direction' command calculate checksum of received command send checksum byte "c"

checksum "c" = 2's-complement [command_letter + w + x + y + z]

Put data pattern

The "P", 'Put data pattern' command sets data on the input contact fingers of the MUT, but only on the contact fingers that have been defined as inputs by a previous "D" command. The command letter is followed by 4 bytes (32 bits) of data that actually define each contact finger according to the "Byte / Contact Finger" table. A '1'-bit sets the corresponding contact finger to TTL high, a logical "1", and a '0'-bit sets the corresponding contact finger to TTL low, a logical "0".
When the Tester receives the "P" command, the Tester responds with a single byte. This byte, called the checksum, is the acknowledgement for the computer that the "P" command is accepted. The byte is the 2's-complement of the addition without overflow of the received command letter and the 4 bytes of data, and as such the computer can check that the command is received correctly. In the following representation "wxyz" represent the 4 bytes of data, "c" the checksum byte.

Computer	flow	Tester
send command "Pwxyz" await response receive checksum and verify	→ ← 	receive 'Put data pattern' command calculate checksum of received command send checksum byte "c"

checksum "c" = 2's-complement [command_letter + w + x + y + z]

Remark. The Tester will accept and reply to the "P" command, but the Tester will not process / execute the command as long as the Tester is in the Safe State, or did not receive a "D" command after the Safe State was left.

Query data pattern

The "Q", 'Query data pattern' command is like the "P" command, in that the "Q" command sets data on the contact fingers of the MUT, but also asks for a result of the data on all the contact fingers. When the Tester receives the "Q" command, the Tester responds with five bytes, which is for the computer the acknowledgement that the "Q" command is accepted. The first four bytes represent 32 bits which correspond with the contact fingers as described in the "Byte / Contact Finger" table.
The last byte, called the checksum, is the 2's-complement of the addition without overflow of the received command letter, the 4 bytes of received data, and the 4 bytes of result data, so that the computer can check that the command and the sent result are communicated correctly.
Each bit in the 4 byte result reflects the state ('0'/ '1') as seen by the Tester hardware. This feature can be used to detect a "stuck" condition on the inputs. (Explanation: when the Tester sets a logic level '0' or '1' on an input, the bit for that input must have that same logic level in the result data from the "Q" command. If the bit in the result has the other logic level, the input is so-called "stuck". As an input of the FlipChip can be stuck to '0' or '1', the software on the PC must test the input with both levels to detect a "stuck" condition).
In the following representation "wxyz" represent the 4 bytes of data, "pqrs" represents the 4 bytes result data, and "c" the checksum of the command letter, 4 bytes of data, and the 4 bytes result data.

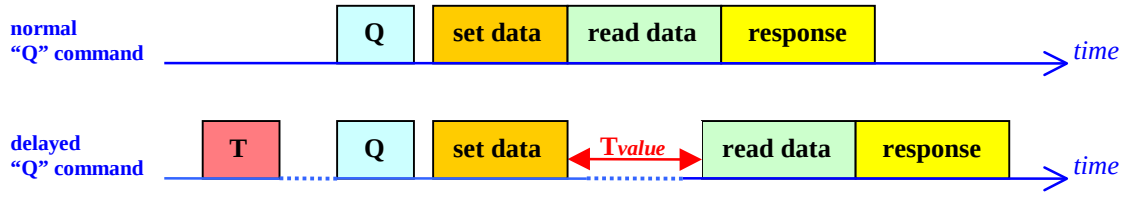
Computer	flow	Tester
send command "Qwxyz" await reply	→	receive 'Query data pattern' command read result data bit information

	←	calculate checksum
receive result 5 data bytes	←	send result data “pqrs” and checksum “c”
verify checksum		

checksum “c” = 2’s-complement [command_letter + w + x + y + z + p + q + r + s]

Time delayed Qread

The “T”, ‘Time delayed Qread’ command generates a delayed read action in the first “Q” command that follows the “T” command. The “Q”, ‘Query data pattern’ sets data on the (input) contact fingers of the MUT, and then reads the data on all the contact fingers, see the description of the “Q” command. The Tester accepts more than one “T” command, but the value of the last received “T” command before a “Q” command is used to insert a delay between the ‘set data’ action on the contact fingers and the ‘read data’ action from the contact fingers. See the figure.
Remark. The “T” command only applies to the first “Q” command that follows that “T” command. If you need more “Q” commands with a delayed read operation, you must specify a “T” command before each of those “Q” commands.



The “T” command letter is followed by 2 bytes (16 bits) of data that actually defines the time delay factor. The 1st byte is the msb part, the 2nd byte is the lsb part. The unit of time delay is 100 µsec, thus the command “T03E8” will insert a delay of 100 msec (1000 x 100 µsec). An example for the use of the “T” command is the test of the output of a one-shot after the one-shot is triggered by the ‘set data’ action of the “Q” command.
When the Tester receives the “T” command, the Tester responds with a single byte. This byte, called the checksum, is the acknowledgement for the computer that the “T” command is accepted. The byte is the 2’s-complement of the addition without overflow of the received command letter and the 2 bytes of data, and as such the computer can check that the command is received correctly. In the following representation “xy” represent the 2 bytes of data, “c” the checksum byte.

Computer	flow	Tester
send command “Txy”	→	receive ‘Time delayed Qread’ command
await response		calculate checksum of received command
	←	send checksum byte “c”
receive checksum and verify		

checksum “c” = 2’s-complement [command_letter + x + y]

Set Tester in Safe State

When the Tester is powered up, it is automatically is in Safe State. Safe State means that the contact fingers of the connector on the Connector Board are not “hot”, they do not put a voltage level on the contact fingers. It is good practice to *start and end* the test session of a FlipChip with the “S” command. When the Tester receives the “S” command, the Tester responds with a single byte. This byte, called the checksum, is the acknowledgement for the computer that the “S” command is accepted. The checksum is the 2’s-complement of the received command, and as such the computer can check that the command is received correctly. In the following representation “s” represents the checksum byte.

Computer	flow	Tester
send command “S”	→	receive ‘Set Tester in Safe state’ command

await response receive checksum and verify	←	disable *all* (32 or 64) contact fingers calculate checksum of received command send checksum byte “s”

Remark. As the command, sent by the computer, is always “S” (hexadecimal 0x53), the checksum for the “Set Tester in Safe State” command is always hexadecimal 0xAD (the 2’s-complement of 0x53).

Enable Tester hardware

Before the Tester will drive inputs for any “P” or “Q” commands, the Tester must have received the “E” command, which enables the Tester hardware. Note that the “D” command is always accepted and processed to define the configuration for the FlipChip to be tested. This means that you can define the input/output contact fingers of the FlipChip *while* the contact fingers themselves remain in the safe state. When the Tester receives the “E” command, the Tester responds with a single byte. This byte, called the checksum, is the acknowledgement for the computer that the “E” command is accepted. The checksum is the 2’s-complement of the received command, and as such the computer can check that the command is received correctly. In the following representation “e” represents the checksum byte.

Computer	flow	Tester
send command “E”	→	receive ‘Enable Tester hardware’ command enable *all* (32 or 64) contact fingers (* note)
await response		calculate checksum of received command
receive checksum and verify	←	send checksum byte “e”

* note). If a “D” command was received between an “S” command and the “E” command, the Enable command will use the “Define I/O direction” command to set the specified contact fingers for the inputs for the FlipChip. If there was no “D” command, all contact fingers are defined as output which means that the Tester will not drive the contact fingers (safe condition). The *safe condition* is not the same as the *Safe State.* In the safe condition the Tester hardware connected to the contact fingers is active, and in Safe State the Tester hardware connected to the contact fingers is in a so-called 3-state, which represent a high impedance (virtual no connection).

Remark 1. The Tester responds on the received “P” and “Q” commands to the computer as described, but will not drive any contact fingers as long as the Tester did not receive the “E” and “D” command.
Remark 2. As the command, sent by the computer, is always “E” (hexadecimal 0x45), the checksum for the “Enable Tester hardware” command is always hexadecimal 0xBB (the 2’s-complement of 0x45).

Specify FlipChip Logic type

The “L”, ‘Specify Logic’ command identifies the hardware logic of the MUT. The command letter is followed by one byte of data that actually identifies the logic type.
When the Tester receives the “L” command, the Tester responds with a single byte. This byte, called the checksum, is the acknowledgement for the computer that the “L” command is accepted. The byte is the 2’s-complement of the addition without overflow of the received command letter and the data byte, and as such the computer can check that the command is received correctly. The defined logic types are the following.

Tester Hardware	‘#’	Type of logic family
Rev.1	ASCII ‘1’ (hex 31)	TTL-based
Rev.2	ASCII ‘1’ (hex 31)	TTL-based
	ASCII ‘2’ (hex 32)	POSIBUS
	ASCII ‘3’ (hex 33)	NEGIBUS

In the following representation “#” represents the logic type, “c” the checksum byte.

Computer	flow	Tester

send command “L#”	→	receive ‘Specify Logic’ command check and set Tester to requested logic
await response		calculate checksum of received command
	←	send checksum byte “c”
receive checksum and verify		

checksum “c” = 2’s-complement [command_letter + #]

Remark.
The ‘Specify Logic’ command is not needed for the Rev.1 Tester, as this Tester implicitly only can handle TTL-based FlipChips. If a Rev.1 Tester receives the ‘Specify Logic’ command, the Tester will check that the specified logic is “TTL”. If the specified logic is not “TTL”, the Tester will return a wrong checksum to indicate that the requested logic is not supported.
The ‘Specify Logic’ command is required for the Rev.2 Tester. If the specified logic is not recognized, the Tester returns a wrong checksum to indicate that the requested logic is not supported. The “wrong” checksum returned is 0x00. See also the description of the ‘Report Tester State’ command.

Report Tester State

The “?”, ‘Report Tester State’ command, asks for the current status of the Tester. The Tester responds with two bytes, which is for the computer the acknowledgement that the “?” command is accepted. The first byte, called the Tester_State indicates whether the power supply is applied to the FlipChip (in row “A”) and the status of the Tester. The Tester_State byte is a set of 8 bits, of which at this moment only 4 bits are used and one bit is reserved for future use, according the following “Tester_State” table. Bit [7] is the most significant bit, and bit [0] is the least significant bit in the Tester_State byte.

bit	***description***	***“0”***	***“1”***
0	Tester Status	SAFE	ACTIVE
1-2	Hardware identification	see Configuration table	
3	*reserved*		
4	FlipChip Power	Power OFF	Power ON
5-7	*not used*		

Remark.
The hardware identification bits indicate to the PC software what Tester configuration is connected. At this moment only “Revision #1” exists, which only supports TTL-based FlipChips. A Tester “Revision #2” is under development. Rev.2 is a more general purpose Tester as it supports TTL, but also POSIBUS and NEGIBUS FlipChips, and all kind of other voltage levels. Bit #1 indicates single/double FlipChip setup, and bit #2 indicates Rev.1/Rev.2 hardware. See the following Configuration table.

Hardware identification	**Tester_State byte**	
	bit [2]	bit [1]
Rev.1 Single-Height FlipChips	**1**	**1**
Rev.1 Double-Height FlipChips	**1**	**0**
Rev.2 Single-Height FlipChips	**0**	**1**
Rev.2 Double-Height FlipChips	**0**	**0**

Configuration table.

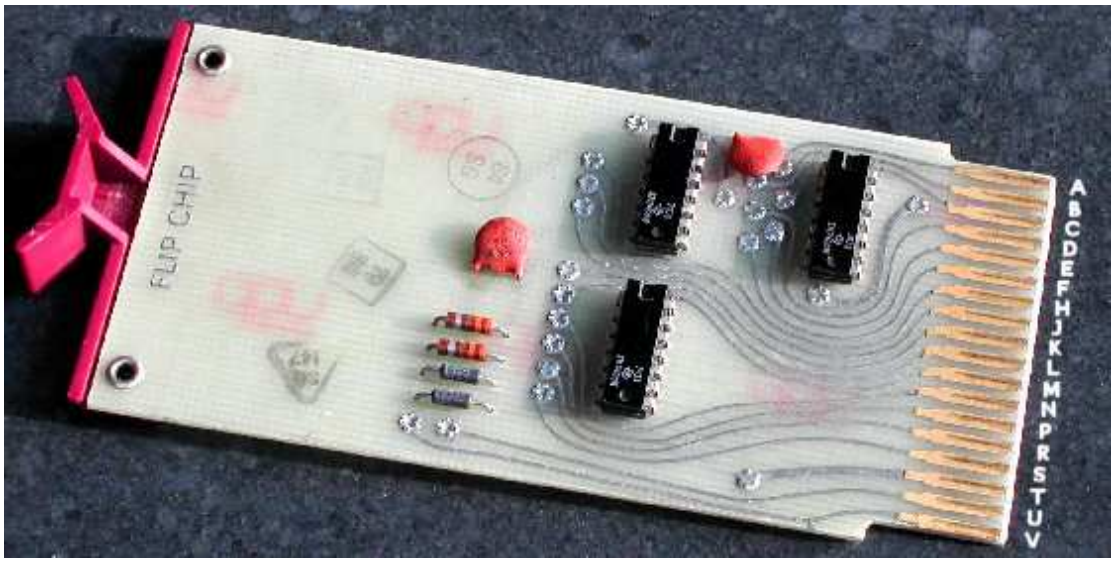
The second byte, called the checksum, is the 2’s-complement of the addition without overflow of the received command letter, and the Tester_State byte, so that the computer can check that the command and the sent result are communicated correctly.
In the following representation “c” represents the checksum byte.

Computer	flow	Tester
send command “?”	→	receive ‘Report Tester State’ command read S/R flip flop
await response		calculate checksum
	←	send Tester State and checksum byte “c”
receive checksum and verify		

checksum “c” = 2’s-complement [command_letter + Tester_State]

“Byte / Contact Finger” table

The data to/from the Tester is always represented by 4 bytes, which represent 32 contact fingers. The 1st byte is (as it says) the first byte sent or received, etc. As defined by DIGITAL, the contact fingers on the component side of the FlipChip have the number “1”, and the contact fingers on the solder side of the FlipChip have the number “2”. If no number is specified in the documentation of a FlipChip, the contact finger at the solder side is meant. See the figure for the contact finger - letter position allocation. When you look at the component side of a FlipChip with the contact fingers pointing towards you, contact finger A is at the right and contact finger V is at the left.



The contact fingers of a single-height FlipChip (view at the component side)

The following table shows the bit-to-contact finger relation.
In each data byte, bit [7] is the most significant bit, and bit [0] is the least significant bit.

[bit]	**1st byte**	**2nd byte**	**3rd byte**	**4th byte**
[7]	A1	K1	V1	M2
[6]	B1	L1	D2	N2
[5]	C1	M1	E2	P2
[4]	D1	N1	F2	R2
[3]	E1	P1	H2	S2
[2]	F1	R1	J2	T2
[1]	H1	S1	K2	U2
[0]	J1	U1	L2	V2

Notes.

1. The contact fingers A2, B2, C2, and T1 have *no* corresponding bit allocated. This applies to *both* the “A” and “B” connectors of the Connector Board.

2. On the “A” connector, the contact fingers A2, B2, C2, and T1 are wired according the standard connection of the FlipChip modules, which is as follows.
   - A2 = +5 Volt power supply
   - B2 = -15 Volt power supply (not connected by the Tester hardware)
   - C2 = GND of the power supply
   - T1 = additional GND connection for double-sided FlipChips

3. On the “B” connector, the contact fingers A2, B2, C2, and T1 should not be connected to the Tester voltages unless the specific module requires them. This usually means you

would open the DIP switches for BA2, BB2, BC2, and BT1. If you must test a double-height FlipChip which uses one or more of the contact fingers A2, B2, C2, and T1 on the "B" connector for a signal, you must connect that contact finger to an unused contact finger. The Tester software on the PC will tell you which connections you must make or break.

Checksum calculation

The Tester calculates a checksum for each command that the Tester receives. The calculated checksum includes the received command letter, if present, the data byte(s) in the received command, and, if present that result data byte(s). The checksum is the 2's-complement of a simple 8-bit addition of all the bytes without taking into account a possible overflow (carry bit). See the following examples.

Example 1: "Set Safe State" command

sent bytes : 0x53

checksum :
2's-complement (0x53) = **0xAD**

Example 2: "Put" command

sent bytes : 0x50 0xE3 0x47 0xA2 0x61

checksum :
2's-complement (0x50 + 0xE3 + 0x47 + 0xA2 + 0x61) =
2's-complement (0x7D) = **0x83**

Example 3: "Report State" command

sent bytes : 0x3F

[result data bytes : 0x11]
checksum :
2's-complement (0x3F + 0x11) = *2's-complement* (0x50) = **0xB0**

Example 4: "Query" command

sent bytes : 0x51 0xE3 0x47 0xA2 0x61

[result data bytes : 0xE7 0x67 0xB2 0x73]
checksum :
2's-complement (0x51 + 0xE3 + 0x47 + 0xA2 + 0x61 + 0xE7 + 0x67 + 0xB2 + 0x73) =
2's-complement (0xF1) = **0x0F**

The computer can verify that no errors occurred during the communication. The computer calculates a simple 8-bit addition of *all* data bytes sent and received for one complete command without taking into account a possible overflow (carry bit). The addition includes the checksum calculated by the Tester. If the communication was without errors, the result of the addition must be hexadecimal 0x00 (8 bit !). See the following example.

Example : "Query" command, result received by the computer

sent bytes : 0x51 0xE3 0x47 0xA2 0x61

[result data bytes : 0xE7 0x67 0xB2 0x73]
checksum :
2's-complement (0x51 + 0xE3 + 0x47 + 0xA2 + 0x61 + 0xE7 + 0x67 + 0xB2 + 0x73) =
2's-complement (0xF1) = **0x0F**

received data : 0xE7 0x67 0xB2 0x73 0x0F
calculated result : (0x51 + 0xE3 + 0x47 + 0xA2 + 0x61 + 0xE7 + 0x67 + 0xB2 + 0x73 + 0x0F)

= 0x00 ***(a correct transmission must have a calculated result of 0)***