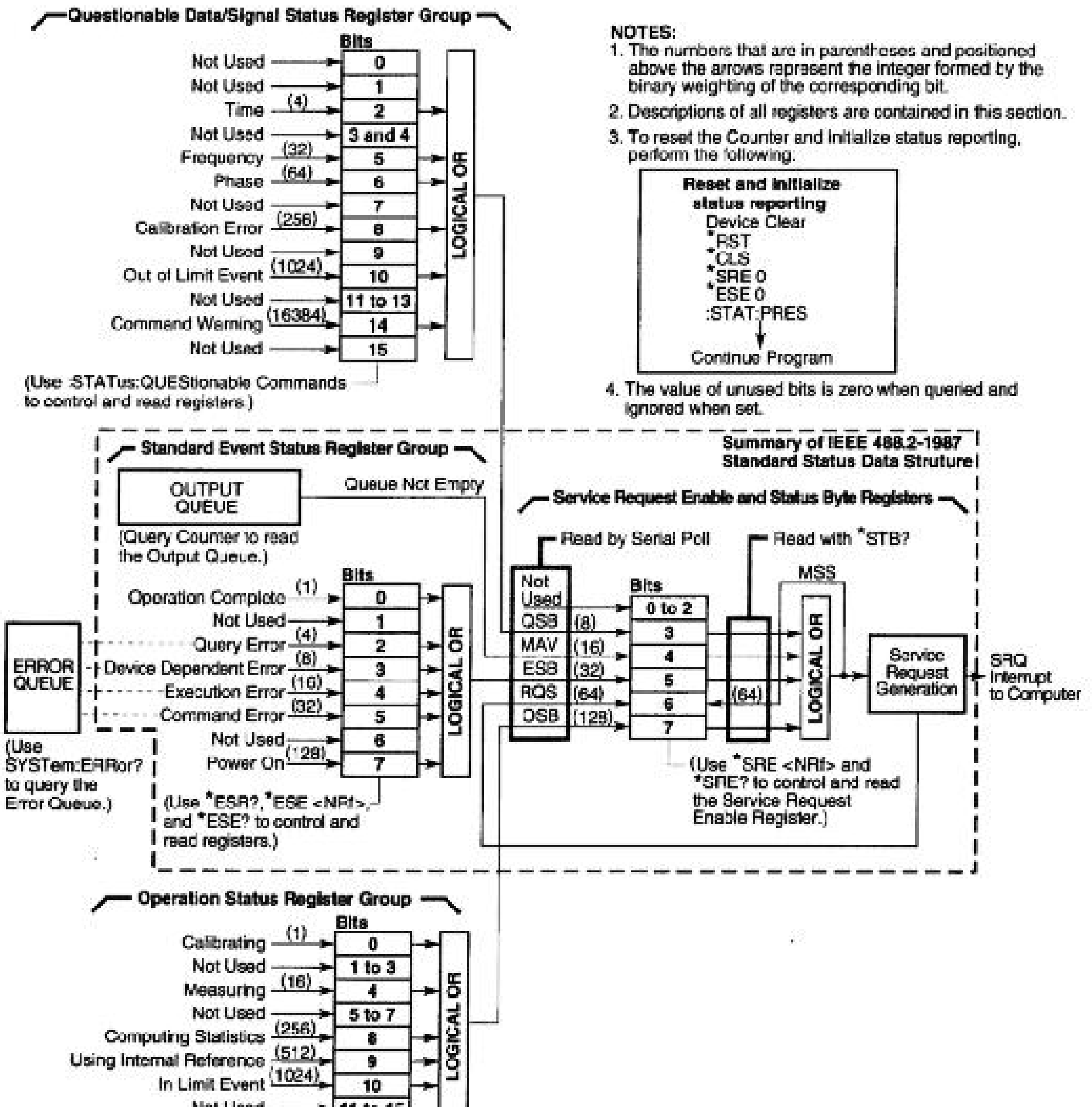
How can I use the Status System to make sure that an operation is complete?

Measuring a low frequency with a reciprocal counter, performing a swept-sine measurement with a dynamic signal analyzer or spectrum analyzer, and scanning through a list of channels with a data acquisition system (to mention a few of the most outstanding examples), all share a common source of trouble. These operations take time. Any operation that follows (such as reading the data resulting from the operations) must be delayed to insure that it doesn't interrupt the ongoing measurement, or cause a timeout. A seemingly reasonable solution would be to introduce delays or wait periods after the offending operation. But this poses a problem: How do you determine the time needed for the operation to complete without adding unnecessary delays to your program? A better solution is found in proper use of the Status System.

Most instruments have a set of registers that report the status of the instrument including various errors. These can be used to assert an SRQ (service request) and thus indicate to the controller that an event of interest has occurred or that certain condition exists.

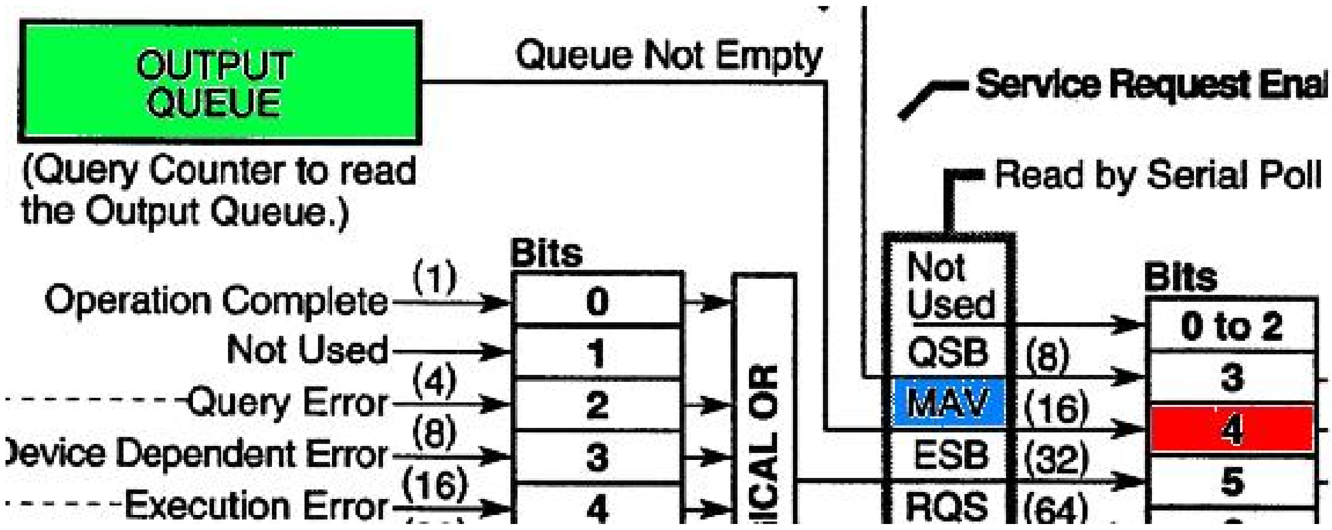
For the purpose of studying the status system, we will use the 53131A counter. However, the concepts are applicable to most instruments. The following is a diagram of the 53131A Status System.


Questionable Data/Signal Status Register Group
Bits
Not Used
Not Used
Time (4)
Not Used
Frequency (32)
Phase (64)
Not Used
Calibration Error (256)
Not Used
Out of Limit Event (1024)
Not Used
Command Warning (16384)
Not Used
0
1
2
3 and 4
5
6
7
8
9
10
11 to 13
14
15
LOGICAL OR
(Use :STATus:QUEStionable Commands to control and read registers.)
NOTES:
1. The numbers that are in parentheses and positioned above the arrows represent the integer formed by the binary weighting of the corresponding bit.
2. Descriptions of all registers are contained in this section.
3. To reset the Counter and Initialize status reporting, perform the following.
Reset and Initialize status reporting
Device Clear
*RST
*CLS
*SRE 0
*ESE 0
:STAT:PRES
Continue Program
4. The value of unused bits is zero when queried and ignored when set.
Summary of IEEE 488.2-1987 Standard Status Data Struture
Standard Event Status Register Group
OUTPUT QUEUE
(Query Counter to read the Output Queue.)
Queue Not Empty
Service Request Enable and Status Byte Registers
Read by Serial Poll
Read with *STB?
MSS
ERROR QUEUE
(Use SYSTem:ERRor? to query the Error Queue.)
Operation Complete (1)
Not Used
Query Error (4)
Device Dependent Error (8)
Execution Error (16)
Command Error (32)
Not Used
Power On (128)
Bits
0
1
2
3
4
5
6
7
LOGICAL OR
(Use *ESR?, *ESE <NRf>, and *ESE? to control and read registers.)
Not Used
QSB (8)
MAV (16)
ESB (32)
RQS (64)
OSB (128)
Bits
0 to 2
3
4
5
6
7
(64)
LOGICAL OR
Service Request Generation
SRQ Interrupt to Computer
(Use *SRE <NRf> and *SRE? to control and read the Service Request Enable Register.)
Operation Status Register Group
Bits
Calibrating (1)
Not Used
Measuring (16)
Not Used
Computing Statistics (256)
Using Internal Reference (512)
In Limit Event (1024)
0
1 to 3
4
5 to 7
8
9
10
LOGICAL OR

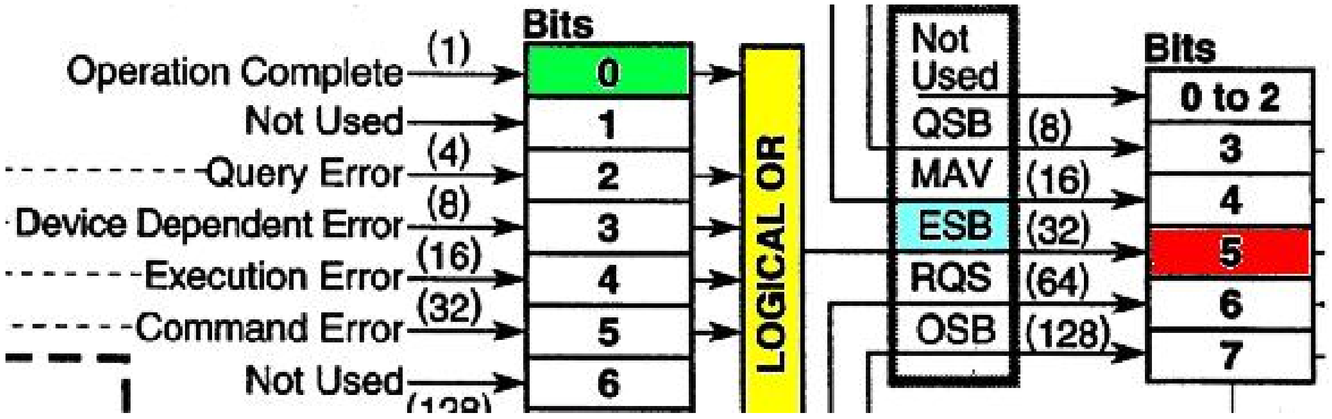
Not Used
11 to 15
(Use :STATus:OPERation Commands to control and read registers.)

Our main interest will be the output queue, the standard event register and the status byte. Before we can program the status register we must first clear it with the command *CLS.

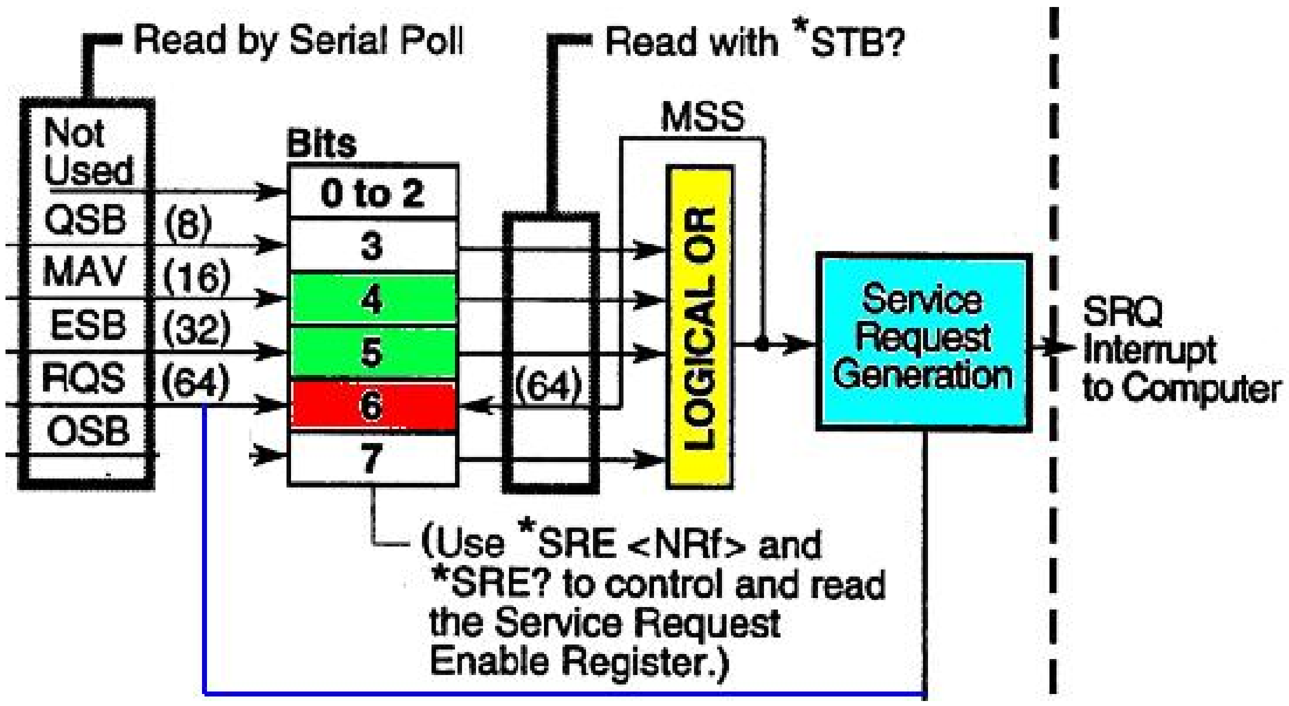
The output queue can have one of two conditions, it can have data in it as a result of a previous operation or not. If there is data in the output queue, this is indicated by setting bit four of the status byte (MAV -- message available).



Each register is paired up with an enable register (not shown in these illustrations). For example, the standard event register has a corresponding standard event enable register. Since our interest is in bit zero of the standard event register, we must enable it by placing a one in the corresponding bit of the standard event enable register. This is done by writing a decimal value equivalent to the binary weight of the bit (in this case 1, because 2^0=1). The command would be *ESE 1. When and if bit zero of the standard event register is set, this bit is "ANDed" with bit zero of the standard event enable register. If both event and enable bits are set, a one is sent to bit five of the status byte. This holds true for any bit of the standard event register, since the bits resulting from "bitwise ANDing" the event register with the enable register are then "ORed".

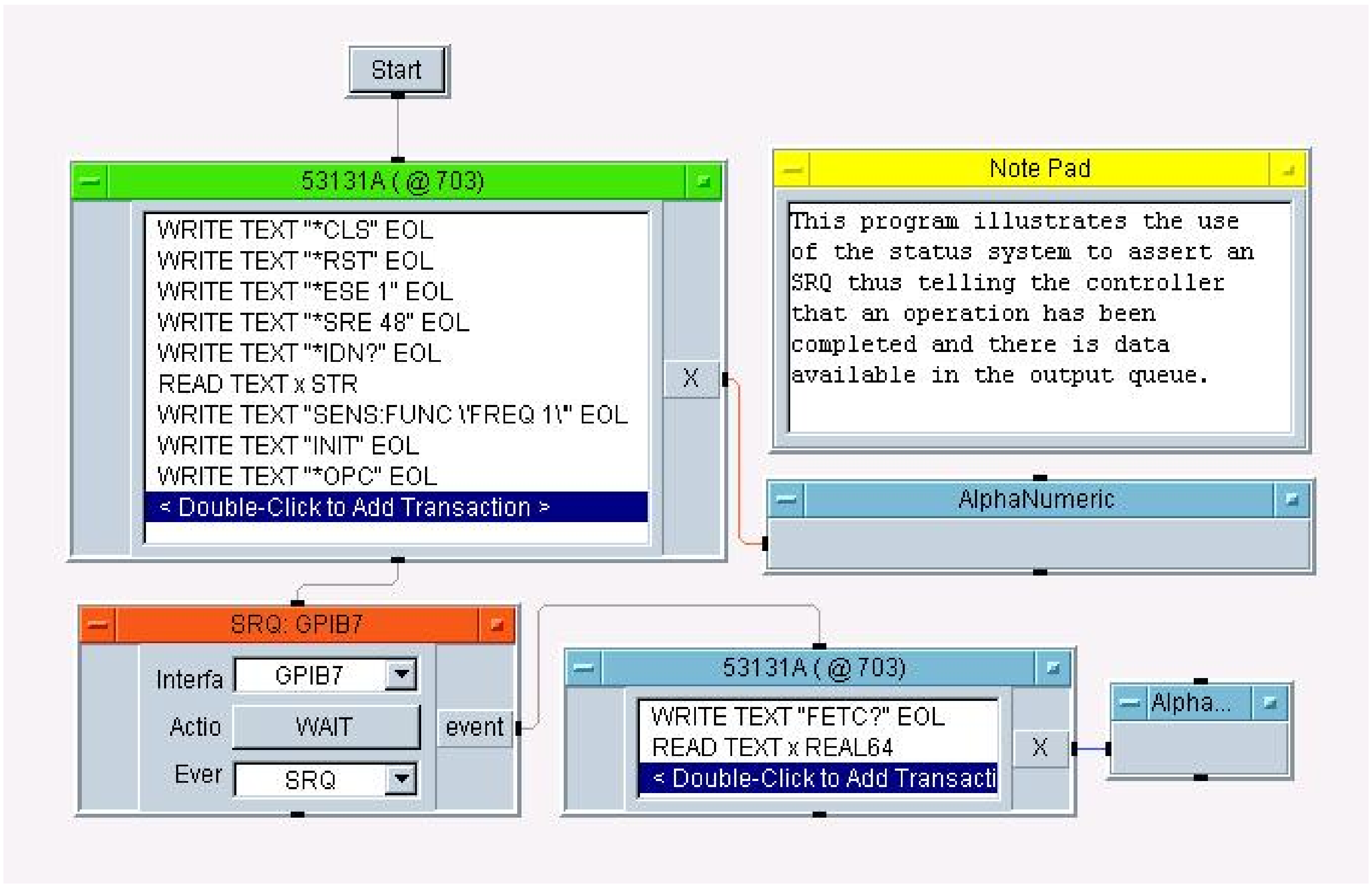


In order to achieve our goal of asserting an SRQ, the Status Byte has to be enabled. We can do this by writing the decimal sum of the binary weight of the bits we want to enable. In this case we wish to enable bits four and five. Bit 4 has a binary weight of 16 (2^4=16) and bit five has a binary weight of 32 (2^5=32). The sum is 48. So we will send the command *SRE 48.


Read by Serial Poll
Read with *STB?
MSS
Not Used
Bits
0 to 2
QSB (8)
3
MAV (16)
4
ESB (32)
5
RQS (64)
6
(64)
OSB
7
LOGICAL OR
Service Request Generation
SRQ Interrupt to Computer
(Use *SRE <NRf> and *SRE? to control and read the Service Request Enable Register.)

When and if bits four and five get set, they will be "ANDed" with the corresponding bits of the status enable register. If any bit of the status register is set and its counterpart in the status enable register is also set, bit six of the status register will be set, causing the GPIB bus to assert an SRQ.

We have not yet discussed how to get the "operation complete" bit (bit 0 of the standard event register) to "set". This cannot happen without some way of identifying the operation of interest. We can do this by sending the command *OPC after the operation of interest has been initiated. The following VEE program illustrates the entire process.



```
WRITE TEXT "*CLS" EOL
WRITE TEXT "*RST" EOL
WRITE TEXT "*ESE 1" EOL
WRITE TEXT "*SRE 48" EOL
WRITE TEXT "*IDN?" EOL
READ TEXT x STR
WRITE TEXT "SENS:FUNC \"FREQ 1\" EOL
WRITE TEXT "INIT" EOL
WRITE TEXT "*OPC" EOL
< Double-Click to Add Transaction >
```

```
This program illustrates the use
of the status system to assert an
SRQ thus telling the controller
that an operation has been
completed and there is data
available in the output queue.
```

Interfa: GPIB7
Actio: WAIT
Ever: SRQ

```
WRITE TEXT "FETC?" EOL
READ TEXT x REAL64
< Double-Click to Add Transacti
```

There is only one thing left to do in order to make this work. The program must have a function that either causes other operations to be delayed until the SRQ is asserted or interrupts other operations when the SRQ is asserted. The above program illustrates the use of an interface event object to cause the program to wait until the SRQ has been asserted.