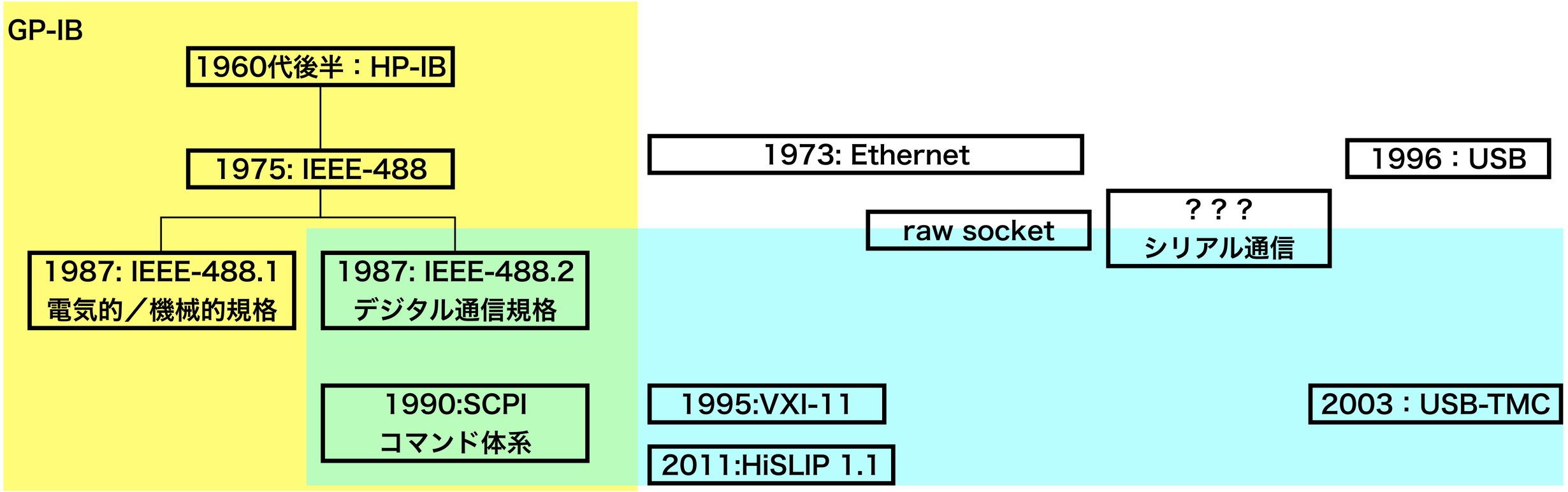
SCPI装置を EPICS-StreamDeviceで 制御する

Noboru Yamamoto (KEK/J-PARC center)

2020.4.16 (last modified 2024.2.16)

SCPI装置について少し

HP-IBからSCPIへ



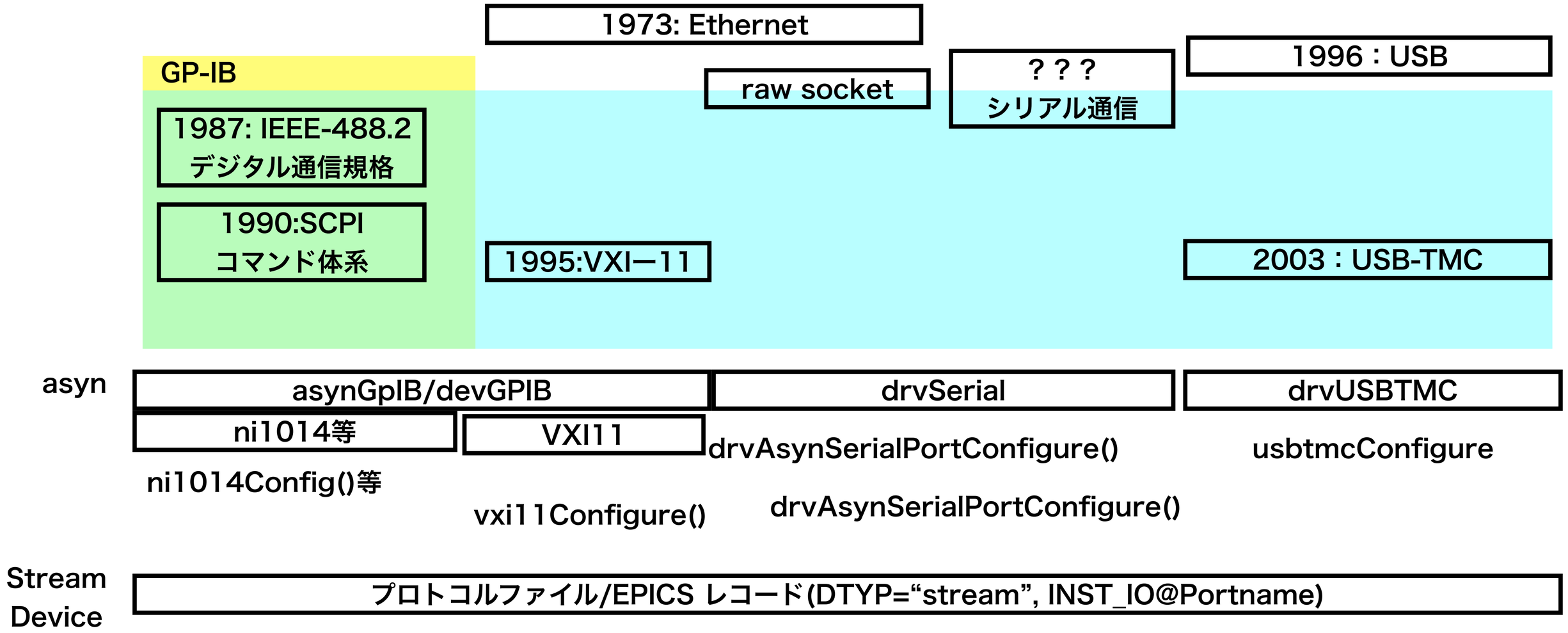
HP-IBの流れを汲む、測定装置などの制御には、

1. 通信路としてEthernet/GPIB(IEEE488)/Seiral/USBなど様々なものが使われている。
2. コマンド体系としてはSCPI (Standard Commands for Programmable Instrumentation)が採用されている。

「SCPI」とは

- SCPI(Standard Commands for Programmable Instruments, often pronounced “skippy”)はオシロスコープなどの測定をPCなどの計算器を使って制御する際のコマンドに関する国際規格の一つです。
  - SCPIは”:WAV:DATA?;”といった**コマンドの文法**と、”*IDN?”などの**標準コマンド**を規定しています。
  - SCPIをサポートする測定器と計算機とは、GP-IB(IEEE488), VXI-11, USBTMC(USB Test and Measurement Class)，Serial，Ethernet(raw soket/telnet), HiSLIPなど様々な方法で接続されます。
  - 通信路が変わっても**同じ測定器であれば同じコマンド**を使って制御できることが、SCPIのメリットです。

SCPI装置と
EPICS asyn／StreamDevice



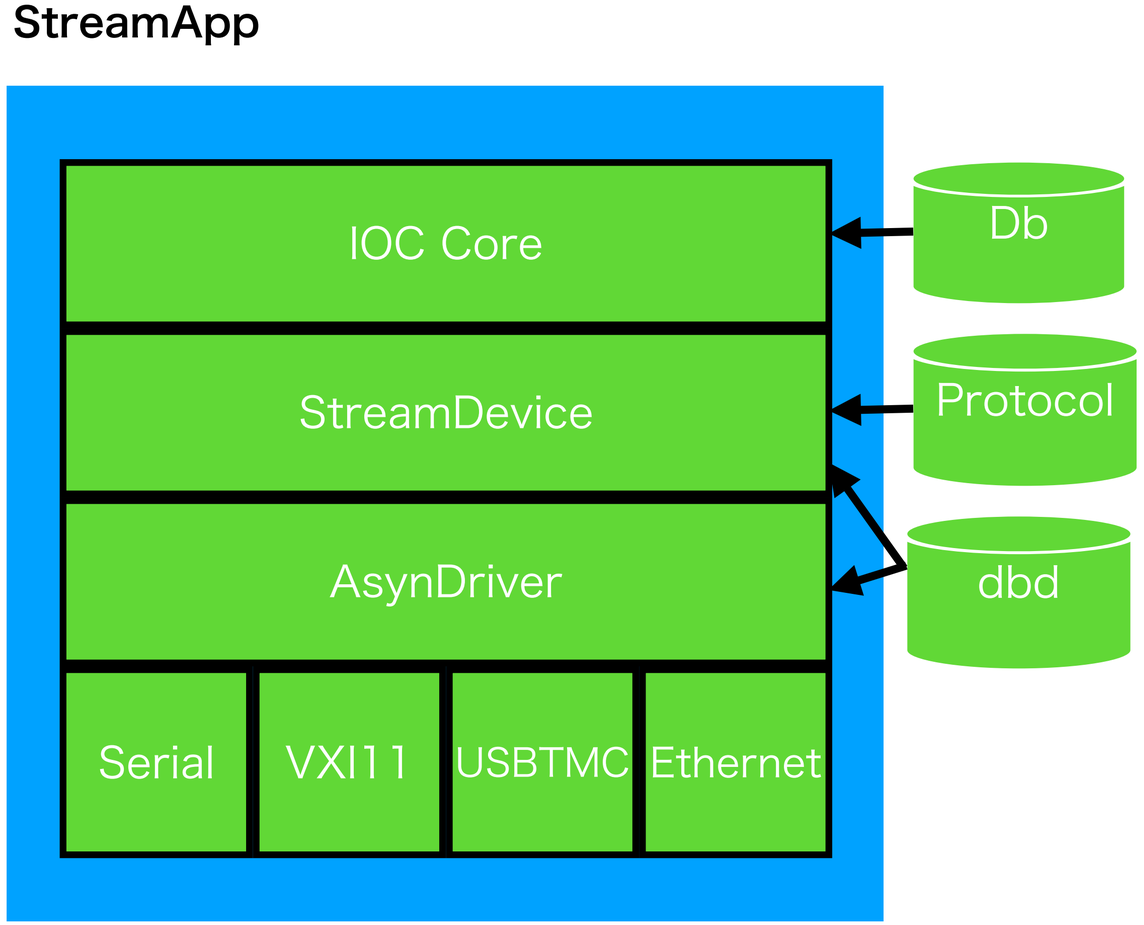
EPICSでの取り扱い

- SCPIをサポートする測定装置はStreamDeviceを使うことで、簡単にEPICS の枠組みを使った制御ソフトに組み込むことができます。

- 測定装置が複数の通信路をサポートする場合には、起動コマンドの中で使う通信路を選択するだけで、データベースやプロトコルファイル(後述)は変更することなく利用することが可能です。

- StreamAppはEPICS-IOCの基本的なソフトウェア(DB, RECORD, CA server)とStreamDeviceおよびそれに付随するソフトウェアが組み込まれたアプリケーションです。

- 測定器を利用するのに必要となるSCPIコマンドを記述したプロトコルファイルとそれらのプロトコルを利用してデータの入出力を実行するレコード定義ファイル(.db)を用意するだけで、装置の基本制御が実現できます。



SRQについて

- SRQ(Service Request)は測定器側から、制御計算機にEventが発生した事を通知する(割り込み）仕組み。
  - 例えば、oscilloscpeがTriggerまちの Arm状態からTriggerを受付、波形データを新規に取得したことをプログラムに通知する。
  - SRQ機能は、HP-IB/GP-IBの時代から存在します。
  - IEEE 488.2でもSRQに関係する***STB/*SRE/*ESE/*ESR/*OPC**などの関係する標準コマンドが規定されています。
- VXI-11, USB-TMCでも**SRQ**の機能がサポートされています。
  - Stream Deviceでは直接この機能を使うことができません。-> 解決方法は後ほど

Asyn／Stream Device の利用法

下準備

- 帯名さんの”Stream Deviceのススメ”を参考に、EPICS Base/sequencer/Asyn driver/Stream Deviceを準備します。

- USBTMCサポートを使用するためには、さらに次のことが必要です。

  1. libusbをOSにインストールする。

  2. StreamDevice/streamApp/asynRegistrars.dbd に
     registrar(usbtmcRegisterCommands)
     を追加した上で、streamAppを再構築する。

makeSuppor.plの利用

- asyn/bin/<arch>/makeSupport.plを利用して、デバイスサポートのテンプレートを簡単に作ることができます。(<arch>はlinux-x86など)
  - 使用例:
    - mkdir YourDevice
    - cd YourDevice
    - <asyn>/bin/<arch>/makeSupport.pl -t streamSCPI myDevice
    - ls myDeviceSup/
      Makefile devmyDevice.db devmyDevice.proto

起動コマンド(st.cmd)

- 使用するAsynドライバの指定はIOCプロセスを起動するst.cmdスクリプト中でそれぞれのasyn port設定コマンドを利用します。デバイスとIOCを接続する方法に合わせた設定(****Configure)を行います。

  - VXI11を利用する場合:
    vxi11Configure(“**OSC**", "169.254.254.254", 0, "0.0", "inst0", 0, 0)

  - TCP socketを利用場合:
    drvAsynIPPortConfigure(“**OSC**", "169.254.254.254:5025", 0, 0, 0);

  - USBTMCを利用する場合:
    usbtmcConfigure("**OSC**", 0x0957, 0x1724)

    - LinuxではIOCプロセスをroot権限で立ち上げる必要があります。

- 接続方法を変更する際には、この例の様に、Asyn port名として同じもの(“**OSC**”)を使うことで、Db側の変更は不要となります。

vxi11 Configure

- vxi11Configure "portName", "host name”, <flags>, "default timeout", "vxiName", priority, <disable auto-connect>
  - portName: asyn port名Dbファイル中で使われる。(文字列)
  - host name: VXI11デバイスのIPアドレス(文字列)
  - <flags>: no SRQ : lock devices : recover with IFCが各ビットに割り当てられている。
  - default timeout: asynが使うデフォルトのtimeout. 浮動小数の秒数を＜文字列＞で設定。
  - vxiName : “inst0”, “gpib0,0”, などのVXI-11デバイス名を文字列で指定。
  - disable auto-connect:自動再接続を行う(0)、しない(1)

drvAsynIPPortConfigure

- drvAsynIPPortConfigure(portName, hostInfo, priority, noAutoConnect, noProcessEos)
  - portName:asyn port名(String)
  - hostInfo: IPアドレス：ポート番号（文字列)
  - priority: asyn プロセスのpriority(整数)
  - noAutoConnect: 1の時自動再接続しない。
  - noProcessEos：Eos(End of String)文字の処理をする(0)しない(1)

usbtmcConfigure

- usbtmcConfigure(port, vendorNum, productNum, serialNumberStr, priority, flags)
  - port: asyn Port 名(string)
  - vendorNum, productNum, serialNumberStr : USBのidVendor, idProduct, serialNumber(文字列）を与える。
    - idVendor, idProductなどはshell コマンドの lsusb を使って入手可能です。(次ページ参照)．serialNumberStr(iSerial)を入手するためには、root権限が必要です serialNumberStrが指定されていない場合は、最初に見つかった装置が使われます。
  - priority: asynプロセスのpriority
  - flags: noAutoconnect flag. 自動再接続を行う（0）、行わない(1)

参考：lsusbの例

```
$ sudo lsusb -v -d 0957:

Bus 003 Device 010: ID 0957:1724 Agilent Technologies, Inc.
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass            0
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x0957 Agilent Technologies, Inc.
  idProduct          0x1724
  bcdDevice            1.00
  iManufacturer           1 Agilent Technologies
  iProduct                2 DSO6014A
  iSerial                 3 MY48260329
  bNumConfigurations      1
```

idVendor:idProduct

serialNumberStr

```
Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength       0x0027
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0xc0
      Self Powered
    MaxPower                0mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           3
      bInterfaceClass       254 Application Specific I
      bInterfaceSubClass      3 Test and Measurement
      bInterfaceProtocol      1 TMC
     <以下略>
```

protocol ファイル

- 使用するSCPIコマンドは、StreamDevice Support の書式に従って、Protocolファイルに記述します。

- StreamDevice Supportが受け付けるProtocolファイルの書式は、帯名さんの説明あるいはStreamDevice Support　ホームページ（http://epics.web.psi.ch/software/streamdevice/）を参考に。

- 以下 Agilent DSO6014Bを試験するために最近書き下ろしたprotocolファイルと対応するデーターベースファイル(.db)から幾つかの例を示します。

protocol/db ファイルの例-1

devDSO6014.proto からの抜粋ー1

protocol名

```
SetPoints{  out ":WAV:POIN %+u;"; }

GetPoints{ out ":WAV:POIN?;"; in "%u"; }
```

devDSO6014.db からの抜粋ー1

```
record(longout, "$(OSC):SET:POINTS"){
    field(DESC, "set a number of data points")
    field(DTYP, "stream")
    field(VAL,  "5000")
    field(PINI, "YES")
    field(OUT,  "@devDSO6014.proto SetPoints $(OSC)")
    field(FLNK,   "$(OSC):GET:POINTS")
}

record(longin, "$(OSC):GET:POINTS"){
    field(DESC, "get a number of data points")
    field(DTYP, "stream")
    field(PINI, "YES")
    field(INP,  "@devDSO6014.proto GetPoints $(OSC)")
}
```

protocol名

asynポート名

protocol/db ファイルの例-2

protocol名

```
SetPointsMode{
    out ":WAV:POIN:MODE %{NORM|MAX|RAW};";
}

GetPointsMode{
    out ":WAV:POIN:MODE?;";
    in "%{NORM|MAX|RAW}";
}
```

devDSO6014.proto からの抜粋ー2

```
record(mbbo, "$(OSC):SET:POINTS:MODE"){
    field(DESC, "set data points mode(NORM/MAX/RAW)")
    field(DTYP, "stream")
    field(PINI, "YES")
    field(VAL, "1")
    field(OUT,  "@devDSO6014.proto SetPointsMode $(OSC)")
    field(FLNK,   "$(OSC):GET:POINTS:MODE")
    field(ZRST, "NORM")
    field(ONST, "MAX")
    field(TWST, "RAW")
}
record(mbbi, "$(OSC):GET:POINTS:MODE"){
    field(DESC, "get a number of data points")
    field(DTYP, "stream")
    field(INP,  "@devDSO6014.proto GetPointsMode $(OSC)")
    field(ZRST, "NORM")
    field(ONST, "MAX")
    field(TWST, "RAW")
}
```

protocol名

asynポート名

devDSO6014.db からの抜粋ー2

protocol/db ファイルの例-3

devDSO6014.proto からの抜粋ー3

```
getWave{
    Separator = "";
    out ":WAV:SOUR CHAN\${1};";
    out ":WAV:FORM WORD;”;
     # Binary(WORD) format for wave data
    out ":WAV:POIN:MODE MAX;";
    out ":WAV:POIN %(\${2}.VAL)+u;";
    out ":WAV:SOUR CHAN\${1};:WAV:DATA?;";
    in "#8%*8c%2r";
}
```

プロトコルの一部をパラメータ化できる。: CHAN\${1}
プロトコル中で、別レコードの値を引用できる。: %(\${2}.VAL)
BinaryのWaveformデータも%r変換を使うことで、
簡単に取り扱える

devDSO6014.db からの抜粋ー3

```
record(waveform, "$(OSC):GET:WAVE:CH1"){
    field(DESC, "get wave data for ch1")
    field(DTYP, "stream")
    field(NELM, "$(NELM)")
    field(FTVL, "USHORT")
    field(INP,
  “@devDSO6014.proto getWave(1,$(OSC):SET:POINTS) $(OSC)")
}
record(fanout, "$(OSC):GET:WAVE:ALL"){
       field(DESC,"Process all 4 records to get waveform")
       field(SELM, "Mask")
       field(SELL, "31")
       field(LNK1, "$(OSC):GET:WAVE:CH1")
       field(LNK2, "$(OSC):GET:WAVE:CH2")
       field(LNK3, "$(OSC):GET:WAVE:CH3")
       field(LNK4, "$(OSC):GET:WAVE:CH4") # timeout!
       field(LNK5, "$(OSC):GET:PRE") # timeout!
}
```

protocol/db ファイルの例-4

devDSO6014.proto からの抜粋ー4

```
getPreamble{
    Separator = "";
    out ":WAV:PRE?;";
    in  "%(\${1}:PRE:FMT)d,",
        "%(\$1:PRE:TYP)d,",
        "%(\$1:PRE:PNT)d,",
        "%(\$1:PRE:CNT)d,",
        "%(\$1:PRE:XINC)f,",
        "%(\$1:PRE:XORG)f,",
        "%(\$1:PRE:XREF)f,",
        "%(\$1:PRE:YINC)f,",
        "%(\$1:PRE:YORG)f,",
        "%(\$1:PRE:YREF)f";
}
```

devDSO6014.db からの抜粋ー4

```
record(stringin, "$(OSC):GET:PRE"){
    field(DESC, "get a preamble for wave data")
    field(DTYP, "stream")
    field(INP,  "@devDSO6014.proto getPreamble($(OSC)) $(OSC)")
}
record(mbbi, "$(OSC):PRE:FMT"){
  field(DESC,"data format")   field(ZRST, "BYTE")
  field(ONST, "WORD")         field(TWST, "ASCII")
}
record(mbbi, "$(OSC):PRE:TYP"){
  field(DESC, "data type”)  field(ZRST, "NORM")
  field(ONST, “PEAK")  field(TWST, "AVER")  field(THST, "HERS")
}
record(longin, "$(OSC):PRE:PNT"){ field(DESC,"number of points")}
record(longin, "$(OSC):PRE:CNT"){ field(DESC, "Average count or 1")}
record(ai, "$(OSC):PRE:XINC"){ field(DESC, "X increment")}
record(ai, "$(OSC):PRE:XORG"){ field(DESC, "X Origin")}
record(ai, "$(OSC):PRE:XREF"){ field(DESC, "X Reference")}
```

EPICS StreamDevice
と
SRQ

SRQへの対応

- IEEE-488.2装置などでSRQによる割り込みを発生できる装置では、測定コマンドを発行した後、SRQが返ってきたところで、波形の取得などを実行することで、全体の効率化が測れると期待できます。
- しかし、StreamDeviceでは ”I/O Intr”は、SRQとは別に、Streamからのデータの応じて、レコードが処理されるメカニズムに使われています。SRQによる割り込み処理には、StreamDeviceとは別の仕組みを利用する必要があります。
- asynDriverレベルでは、GPIB/VXI11/USBTMCなどでSRQを”I/O Intr”として取り扱える。ということで、この部分だけを(streamDeviceではなく） asyn Driverに任せます。
- USBTMCとVXI-11/GPIBで取り扱いが少し変わりますが、動作することを確認しています。
  - ~~VXI11のライブラリにtirpcを使っているシステム(例えばCentOS8)でもVXI11経由にSRQは動作しない。~~
  - GP-IBについてはInterfaceがないので、実際に動作を確認した訳ではありません。

VXI-11/GPIB

- asynドライバレベルではこれらのデバイスはそれ以前に使われていたGPIBデバイスサポートの仕組みを利用しています。

- LinkタイプとしてGPIB_IOをサポートするdevice Supportプログラムが必要です。しかし、SCPIコマンドによるやり取り(デバイスメッセージ）をStream Deviceに任せることにすれば、このデバイスサポートはマルチライン・インターフェイス・メッセージ だけを取り扱うので、機器依存のないデバイスサポートとなります。

    - SRQをサポートするのに必要なコマンドだけを持つGPIB装置device supportを一度用意しておけば、どんな　GPIB/VIX11のデバイスでも利用可能です。

SRQGpib

- 以下に示すgpibCmdsテーブルを持つGPIB デバイスサポートを作成し、Strem Appと同時にIOCにインストールします。

```
static struct gpibCmd gpibCmds[] = {
    /* Param 0, SRQ handler:SQRを受け取るためのレコード*/
    {&DSET_LI, GPIBSRQHANDLER, IB_Q_HIGH, 0,  0, 0, 0, 0, 0, 0, 0, 0, 0},
    /* Param 1, Enable SRQ  SRQ: boSRQonOFFはasyn/devGpibで定義されている*/
    {&DSET_BO, GPIBCVTIO, IB_Q_HIGH, NULL, NULL, 0, 0, boSRQonOff, 0, 0, NULL,NULL,NULL },
   /* Param 2, Read Status Byte: Stream Deviceでも対応可能 */
    {&DSET_AI, GPIBREAD, IB_Q_LOW, "*STB?;", "%d", 0, 32, 0, 0, 0, NULL, NULL, 0}
};
```

- devSRQGpib.dbdには次の三行を記述します。

```
device(longin,    GPIB_IO, devLiSRQGpib,    “SRQGpib")
device(bo,        GPIB_IO, devBoSRQGpib,    "SRQGpib")
device(ai,        GPIB_IO, devAiSRQGpib,    "SRQGpib")
```

SRTGpib.db

SRQEventを受信するためのレコードを用意します。
SRQ発生時に処理を開始するためのレコードへのForward Linkを
設定しておきます。
PORTには使用するポートの名前を与えます。このPORT名はGPIB_IO による
制限のため，　”L <n>”、(<n>は数字0,1,...)の形式だけが有効です。
このレコードの値は、最後のSRQ発生直後のSTBの値です。

```
record(longin,"$(P)$(R)_P$(PORT)_srqHandler") {
    field(DTYP,"SRQGpib")
    field(INP,"#$(PORT) A$(A) @0")
    field(FLNK,”Record_XXXXXX”)
}
```

StremAppへのdevSRQGpibの追加

- StreamAppと組み合わせて使うためにStreamAppの実行ファイルにdevSRQGpibのライブラリをインストールします。

- StreamApp/Makefile に次の行を追加します。

```
streamApp_DBD += devSRQGpib.dbd
PROD_LIBS += devSRQGpib
```

- IOCの起動ファイルに先程の　devSRQGpib.db　をロードするdbLoadRecordsを追加します。

USBTMCとSRQ

- SCPI装置で使われる通信路の一つがUSB-TMC(Test and Measurement Class)です。USB-TMCでもSRQがサポートされています。streamDevice/asyn driverはUSB-TMCをサポートしていますので、VXI11やSerial通信と同じ様にSCPI装置をIOCに組み込むことができます。

- asyn-driverでのUSB-TMCの取り扱いは，　GP-IBやVXI11とは異なりdevGPIBの形式をとっていません。シリアル接続の取り扱いに近く、asynInt32などのデバイスサポートを直接使うことになります。

USBTMCのStreamAppへの組み込み

- 標準的なStreamAppの設定ではUSBTMCのサポートはStreamAppに組み込まれません。
- asynのCONFIG_SITEでDRV_USBTMCを”YES”に設定し、libasynを構築します。
- StreamAppのなかで、以下の変更を追加します。
  - asynRegistrars.dbdに`registrar(usbtmcRegisterCommands)`を追加。
  - Makefieに `streamApp_DBD += asyn.dbd` を追加
  - 起動スクリプトを次のレコードをロードする様に追加。

SRQ:USBTMC

USBTMCのSRQを拾うために、次のレコードを追加するします。DTYPが”asynInt32”であることに注意してください。FLNKの先をStreamDeviceのレコードとすることで、処理を継続します。

```
record(longin, "$(OSC):ASYN:SRQ")
{
    field(DESC, "Device status byte from Service Request")
    field(DTYP, "asynInt32")
    field(INP,  “@asyn($(PORT), 0, 0) SRQ")
    field(SCAN, "I/O Intr")
    field(FLNK, "$(OSC):GET:WAVE:ALL.PROC")
}
```

まとめ

- EPICS/StreamDeviceを使うことで、通信路の選択にかかわらずほぼ同じEPICS IOCを簡単に構築できる。

- この様に、StreamDeviceは強力な方法ですが、VXI-11(やGPIB)が提供するSRQ割り込みを直接使うことができません。AsynDriverのレベルでは、SRQをサポートしているので、SRQ割り込みだけはAsynDriverを使うことで、効率的な運用が可能になります。USBTMC/VXI11ともSRQを受け取るレコードをasynDriver側で作成すれば、可能であることが確認されています。

- VXI11のSRQサポートの動作確認を行ったのはRaspbianおよびMacOS(Catalina 10.15. 4)です。

- CentOSなどFirewallによってSCPI側からhost側のIntr_channelを受け取るRPC serverへの通信が禁止されている場合には、この仕組みは働きません。Port番号がdynamicに変わるので、その度にSCPI側からの該当ポートへの通信を許可する仕組みが必要です。(firewall-cmdで実施できるのですが、汎用的な方法があるかどうか？)

おまけ

SRQ関係のレジスタ

- IEEE488.2に従う装置では、Status Byte Reigster(*STB?)の各ビットにSRQ Enable Registerをマスクとしてかけた結果の全ビットの和が0でない時、SRQを発生します。

- この仕組みは測定器によらず共通ですが、レジスタのビット毎の意味や、それらのビットがAssertされるメカニズムは測定器毎に若干異なります。SRQの使用に際しては、各装置のマニュアルをよく調べておくことが必要です。Agilent(Keysight)とTextronixでのSRQ周りのレジスタの構成を次ページに示しておきます。

SRQ 関連のレジスタ

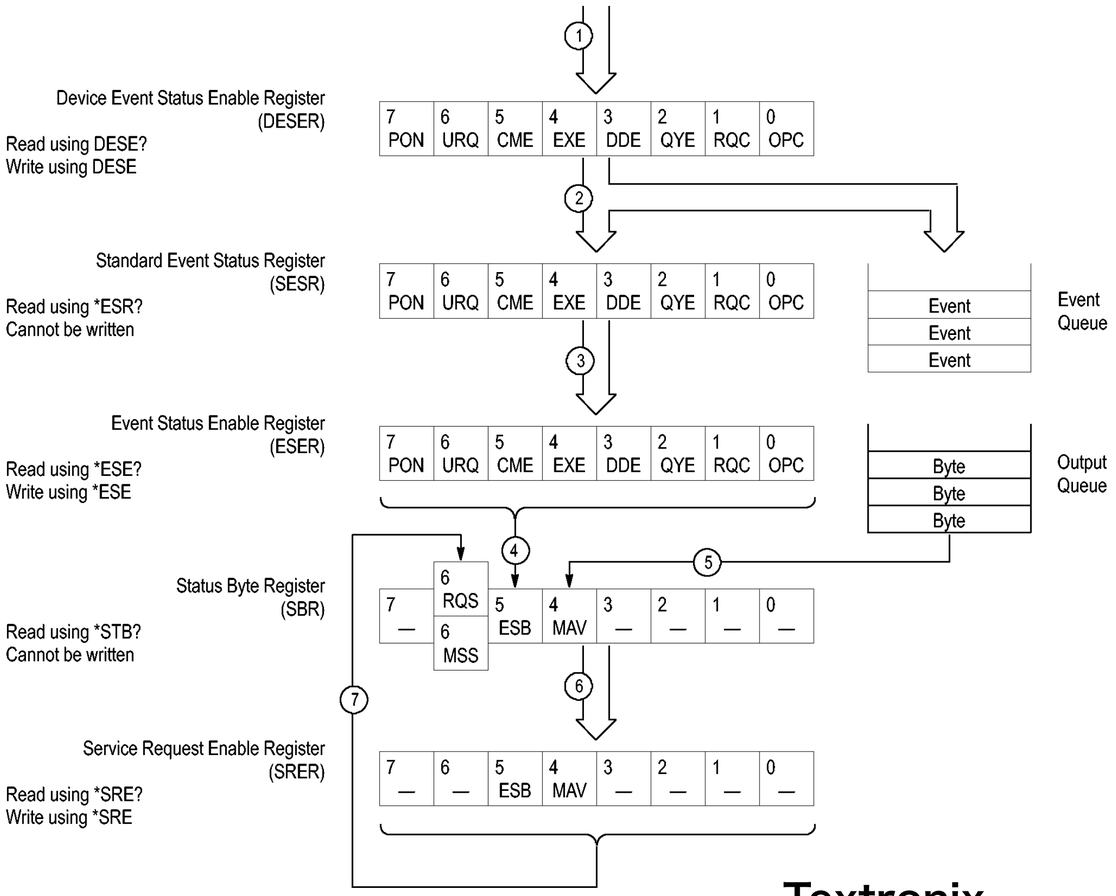
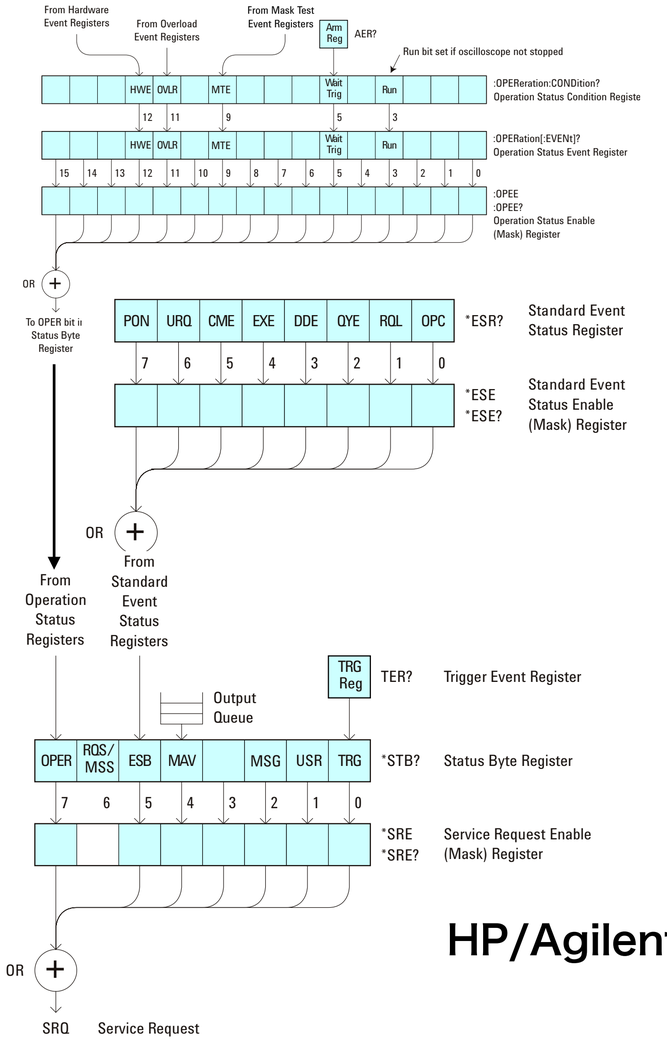


Figure 3-6: Status and Event Handling Process

Textronix



HP/Agilent/Keysight

TextronixとAgilentでのSRQレジスタ周りの違いに注意

Agilent SCPI Socket

raw socketを使った通信の場合、本文で述べたようなSRQを取り扱う一般的な方法はありません。以下のAgilentのマニュアルの記述にある様な方法でSRQが通知される場合には、Stream Deviceで取り扱えます。+nnはSTBの値です。

- 3.2. Service Request

- The user would enable service requests exactly the same as for GPIB. Once service requests have been enabled just listen on the control connection. When service requests have been encountered the instrument will send the string “SRQ +nn\n” to the client. The nn is the status byte for the client to use to determine the source of the service request.