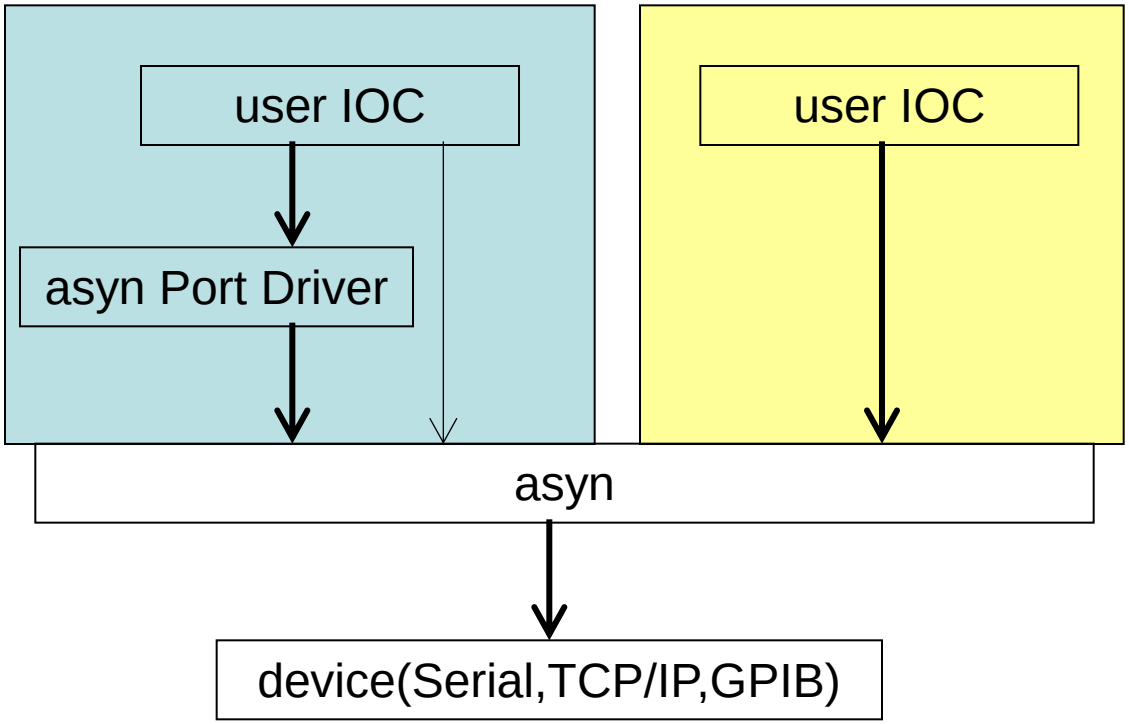
asynPortDriver

C++ base class for asynPortDriver

asynPortDriverの位置付け

- asynPortDriverはasynのC++wrapper



asynPortDriverを使ったほうが良いパターン (StreamDeviceでは難しいデータ)

- 送受信プロトコルがbinaryのみで複雑
- 受信文字列フォーマットが一定でない

```
> MEAS
> RD
> A 1000.0
> RD
> B 20000.0
```

- 1回の通信で複数データを受信

```
> MEAS
> RD
> ch1: 1000.0 ch2: 20.0 ch3: 300.3 ch4: 4000.4
```

- ASCII受信データが複数行で複雑なフォーマット

```
> MEAS
> RD
> ch1 1000.0 V OK
ch2 20.0 degC ALARM
ch3 300.0 mV WARN
ch4 4000.4 mA
```

asynPortDriverの利点

- Device Supportはasynが担当
  - 再接続やタイムアウトはasynが行う
- asyn単体での記述よりは多少コードが少なくて済む

asynPortDriverの欠点

- C++で記述
  - 慣れが必要
- StreamDeviceとの混在が不可
  - できそうなんだけど、、、

asynPortDriverでIOCを作ったデバイス

- 東洋メディック Model 451B(Fluke OEM)

- MOXA シリアルEtherコンバータにてRS-232C接続
- はじめはpythonでcaputしていた。だが、再起動時に色々と問題があるのでStreamDeviceで一度作成したが、”50 uSv/h”の部分を分離できず、どうしても”50”というデータが入ってしまう。そのため、asynPortDriverでIOCを作成した。

```
CMD? T
50 uSv/h
02.6 01.8 01.3 00.9 00.6 00.4 00.2
50 uSv/h
00.0 00.0 00.0 00.0 00.1 00.1 00.1 00.1 00.1 00.2 00.2 00.2 00.2 00.2 00.2
50 uSv/h
00.2 00.2 00.2 00.1 00.2 00.2 00.2 00.2 00.2 00.2 00.2 00.2 00.2 00.2 00.2
50 uSv/h
 :
```

Header File

・コマンド名の定義
・使用関数の設定

```
#define P_StartStop_Str         "START_STOP"                    /* asynInt32,       w  */
#define P_GetData_Str           "GET_DATA"                      /* asynFloat64      r  */

class drvFluke450 : public asynPortDriver {
 public:
  drvFluke450(const char *portName, const char *asynIpPortName);

  /* These are the methods that we override from asynPortDriver */
  virtual asynStatus writeInt32(asynUser *pasynUser, epicsInt32 value);
  virtual asynStatus readFloat64(asynUser *pasynUser, epicsFloat64 *value);

  void readTask(void);

  protected:
      /** Values used for pasynUser->reason, and indexes into the parameter library. */
      int P_StartStop;
#define FIRST_Fluke450_COMMAND P_StartStop
      int P_GetData;
#define LAST_Fluke450_COMMAND P_GetData
  private:

   asynUser      *pasynUserDriver;
   asynStatus    WriteRead(char *sendBuffer, char *recvBuffer, int &recvBufSize, double tmo=TIMEOUT);
   asynStatus    Read(char *recvBuffer, int &recvBufSize, double tmo=TIMEOUT);

   asynStatus    GetData(asynUser *pasynUser, epicsFloat64 &value);
   asynStatus    SetStartStop(asynUser *pasynUser, epicsInt32 value);
};

#define NUM_Fluke450_PARAMS (&LAST_Fluke450_COMMAND - &FIRST_Fluke450_COMMAND + 1)
```

Override可能な関数

```
/* ai */
virtual asynStatus readFloat64(asynUser *pasynUser, epicsFloat64 *value);

/* waveform, aai */
virtual asynStatus readFloat64Array(asynUser *pasynUser, epicsFloat64 *value,
                                    size_t nElements, size_t *nIn);
/* ai, bi, mbbi, longin */
virtual asynStatus readInt32(asynUser *pasynUser, epicsInt32 *value);

/* stringin, waveform, aai */
virtual asynStatus readInt8Array(asynUser *pasynUser, epicsInt8 *value,
                                 size_t nElements, size_t *nIn);
/* waveform, aai */
virtual asynStatus readFloat64Array(asynUser *pasynUser, epicsFloat64 *value,
                                    size_t nElements, size_t *nIn);
/* mbbi */
virtual asynStatus readEnum(asynUser *pasynUser, char *strings[], int values[],
                            int severities[], size_t nElements, size_t *nIn);
/* bo, mbbo, longout */
virtual asynStatus writeInt32(asynUser *pasynUser, epicsInt32 value);

/* ao */
virtual asynStatus writeFloat64(asynUser *pasynUser, epicsFloat64 value);

/* stringout, aao */
virtual asynStatus writeInt8Array(asynUser *pasynUser, epicsInt8 *value,
                                  size_t Elements);
```

Source File(1) コンストラクタ

・createParam関数で関数名と番号、型を関連付け
・asynとの接続
・その他初期化

```
drvFluke450::drvFluke450(const char *portName, const char *asynIpPortName) : asynPortDriver(portName,
                    0, /* maxAddr */
                    NUM_Fluke450_PARAMS,
                    asynInt32Mask | asynFloat64Mask | asynEnumMask | asynDrvUserMask | asynOctetMask, /* Inter face mask */
                    asynOctetMask | asynEnumMask | asynDrvUserMask ,  /* Interrupt mask */
                    0, /* asynFlags.  This driver does not block and it is not multi-device, so flag is 0 */
                    1, /* Autoconnect */
                    0, /* Default priority */
                    0) /* Default stack size*/
{
  char tmpStr[32];
  asynStatus status;

  pasynOctetSyncIO->connect(asynIpPortName, 0, &(this->pasynUserDriver), NULL);

  eventId_ = epicsEventCreate(epicsEventEmpty);
  createParam(P_StartStop_Str,          asynParamInt32,         &P_StartStop);
  createParam(P_GetData_Str,            asynParamFloat64,       &P_GetData);

  /* Create the thread that computes the waveforms in the background */
  status = (asynStatus)(epicsThreadCreate("drvFluke450AsynPortDriverTask",
                                          epicsThreadPriorityMedium,
                                          epicsThreadGetStackSize(epicsThreadStackMedium),
                                          (EPICSTHREADFUNC)::readTask,
                                          this) == NULL);
  if (status) {
    //    printf("%s:%s: epicsThreadCreate failure¥n", driverName, functionName);
  }
}
```

Source File(2)　config

・接続するasynポート名を取得するために必須

```
/* Configuration routine.  Called directly, or from the iocsh function below */
extern "C" {
  /** EPICS iocsh callable function to call constructor for the drvFluke450 class.
   * ¥param[in] portName The name of the asyn port driver to be created.
   * ¥param[in] */
  int drvFluke450Configure(const char *portName, const char *asynIpPortName)
  {
    new drvFluke450(portName, asynIpPortName);
    return(asynSuccess);
  }
  /* EPICS iocsh shell commands */
  static const iocshArg initArg0 = { "portName",iocshArgString};
  static const iocshArg initArg1 = { "asynIpPortName",iocshArgString};
  static const iocshArg * const initArgs[] = {&initArg0, &initArg1};
  static const iocshFuncDef initFuncDef = {"drvFluke450Configure",2,initArgs};
  static void initCallFunc(const iocshArgBuf *args){
    drvFluke450Configure(args[0].sval, args[1].sval);
  }

  void drvFluke450Register(void) {
    iocshRegister(&initFuncDef,initCallFunc);
  }
  epicsExportRegistrar(drvFluke450Register);
}
```

Source File(3) readFloat64

・readFloat64をoverride

```
asynStatus
drvFluke450::readFloat64(asynUser *pasynUser, epicsFloat64 *value)
{
  int function = pasynUser->reason;
  asynStatus status = asynSuccess;
  const char *paramName;
  const char* functionName = "readFloat64";
  int ch = 0;

  /* Fetch the parameter string name for possible use in debugging */
  getParamName(function, &paramName);

  char tmpParamName[256];
  memset(tmpParamName, 0x00, sizeof(tmpParamName));
  strcpy(tmpParamName, paramName);

   if(function == P_GetData) {
    status = this->GetData(pasynUser, *value);
  }
  /* Do callbacks so higher layers see any changes */
  status = (asynStatus) callParamCallbacks();

  if (status)
    epicsSnprintf(pasynUser->errorMessage, pasynUser->errorMessageSize,
                  "%s:%s: status=%d, function=%d, name=%s, value=%f",
                  driverName, functionName, status, function, paramName, *value);
  else
    asynPrint(pasynUser, ASYN_TRACEIO_DRIVER,
              "%s:%s: function=%d, name=%s, value=%f¥n",
              driverName, functionName, function, paramName, *value);
  return status;
}
```

Source File(4) writeInt32

・writeInt32をoverride

```
asynStatus
drvFluke450::writeInt32(asynUser *pasynUser, epicsInt32 value)
{
  int function = pasynUser->reason;
  asynStatus status = asynSuccess;
  const char *paramName;
  const char* functionName = "writeInt32";

  /* Set the parameter in the parameter library. */
  status = (asynStatus) setIntegerParam(function, value);

  /* Fetch the parameter string name for possible use in debugging */
  getParamName(function, &paramName);

  // 通信するコマンド
  if(function == P_StartStop) {
    status = this->SetStartStop(pasynUser, value);
  }

  /* Do callbacks so higher layers see any changes */
  status = (asynStatus) callParamCallbacks();

  if (status)
    epicsSnprintf(pasynUser->errorMessage, pasynUser->errorMessageSize,
                  "%s:%s: status=%d, function=%d, name=%s, value=%d",
                  driverName, functionName, status, function, paramName, value);
  else
    asynPrint(pasynUser, ASYN_TRACEIO_DRIVER,
              "%s:%s: function=%d, name=%s, value=%d¥n",
              driverName, functionName, function, paramName, value);
  return status;
}
```

Source File(5) write/read

・デバイスへのアクセスはasynと同じ

```
asynStatus
drvFluke450::WriteRead(char *sendBuffer, char *recvBuffer, int &recvBufSize)
{
  size_t nActual = 0;
  size_t nRead = 0;
  int    eomReason;
  char   *pValue      = sendBuffer;
  size_t nChars       = strlen(sendBuffer);
  char   *pReadBuffer = recvBuffer;

  asynStatus status = pasynOctetSyncIO->writeRead(pasynUserDriver, pValue, nChars,
                                                  pReadBuffer, MAX_BUF_SIZE, TIMEOUT,
                                                  &nActual, &nRead, &eomReason);
  status = (asynStatus) callParamCallbacks();
  recvBufSize = nRead;
  return status;
}

asynStatus
drvFluke450 ::Read(char *recvBuffer, int &recvBufSize)
{
  size_t nRead = 0;
  int    eomReason;
  char   *pReadBuffer = recvBuffer;

  memset(pReadBuffer, 0x00, sizeof(pReadBuffer));

  asynStatus status = pasynOctetSyncIO->read(pasynUserDriver, pReadBuffer,
                                             MAX_BUF_SIZE, TIMEOUT,
                                             &nRead, &eomReason);
  status = (asynStatus) callParamCallbacks();
  return status;
}
```

db file

・DTYPを”asyn*”に
・OUT/INPに”@asyn($(PORT),$(ADDR),$(TMO)func”

```
record(bo, "$(P):$(R):STARTSTOP") {
    field (DTYP, "asynInt32")
    field (OUT,  "@asyn($(PORT),$(ADDR),$(TMO))START_STOP")
    field (ZNAM, "START")
    field (ONAM, "STOP")
}

record (ai, "$(P):$(R):RADMON") {
    field (DTYP, "asynFloat64")
    field (INP,  "@asyn($(PORT),$(ADDR),$(TMO))GET_DATA")
    field (SCAN, "$(S)")
    field (PREC, "1")
}
```

dbd file

```
include "base.dbd"
include "asyn.dbd"
registrar("drvFluke450Register")
registrar("drvAsynIPPortRegisterCommands")
```

Makefile

```
LIBRARY_IOC += Fluke450Support
Fluke450Support_LIBS += asyn
Fluke450_DBD += drvAsynIPPort.dbd
Fluke450_LIBS += Fluke450Support asyn
```

st.cmd

```
#!../../bin/linux-x86_64/Fluke450

## You may have to change Fluke450 to something else
## everywhere it appears in this file
< envPaths

cd ${TOP}

## Register all support components
dbLoadDatabase("dbd/Fluke450.dbd")
Fluke450_registerRecordDeviceDriver(pdbbase)

epicsEnvSet("BUS",   "DEV1")
epicsEnvSet("DEV",   "RadMon")
epicsEnvSet("USER",  "PFTest")
epicsEnvSet("SCAN",  "1 second")
epicsEnvSet("MVBUS", "apd")

drvAsynIPPortConfigure($(BUS), "xxx.xxx.xxx.xxx:nnnnn", 0, 0, 0)
drvFluke450Configure($(MVBUS), $(BUS))

## Load record instances
dbLoadRecords("db/Fluke450.db","P=$(USER),R=$(DEV),S=$(SCAN),PORT=$(MVBUS),ADDR=0,TMO=1")

#asynSetTraceIOMask("DEV1",0,4)
#asynSetTraceMask("DEV1",0,9)

cd ${TOP}/iocBoot/${IOC}
iocInit()
```