Autoparam, a generic asyn port driver with dynamic parameters

jure.varlec@cosylab.com

EPICS Collaboration Meeting 2022

What is "generic" device support?

- A generic device support module
  - can integrate very different devices
  - without changing/rebuilding the driver
  - when devices speak a common protocol.
- Examples: s7plc, Modbus, Streamdevice …
- Modules for well known protocols are available; what about less common ones?
- autoparamDriver is a base class that makes writing a generic module easier.

Read op in plain asynPortDriver

```
asynStatus testAsynPortDriver::writeInt32(asynUser *pasynUser, epicsInt32 value)
{
    int function = pasynUser->reason;

    if (function == P_Run) {
        /* If run was set then wake up the simulation task */
        if (value) epicsEventSignal(eventId_);
    }
    else if (function == P_VertGainSelect) {
        setVertGain();
    }
    else if (function == P_VoltsPerDivSelect) {
        setVoltsPerDiv();
    }
    ...
```

s7plc module

```
record (ai, "ai-float-1") {
  field (DTYP, "S7plc")
  field (INP,  "@Testsystem0/16 T=FLOAT")
  field (PREC, "6")
  field (SCAN, "I/O Intr")
}
```

modbus module

```
record(longin, "$(PREFIX)Ser") {
    field(DESC, "Serial Number")
    field(SCAN, "I/O Intr")
    field(DTYP, "asynInt32")
    field(INP, "@asyn($(PORT),42)UINT16")
}
```

ADS module

```
record(longin, "$(P):sym_int32_counter") {
    field(DESC, "Read ADS by symbolic name")
    field(DTYP, "asynInt32")
    field(INP,  "@asyn($(PORT)) DINT R P=$(ADS_PORT) V=TestPlan.sym_int32_counter")
    field(SCAN, "I/O Intr")
}
```

s7plc module

```
record (ai, "ai-float-1") {
  field (DTYP, "S7plc")
  field (INP,  "@Testsystem0/16 T=FLOAT")
  field (PREC, "6")
  field (SCAN, "I/O Intr")
}
```

modbus module

```
record(longin, "$(PREFIX)Ser") {
    field(DESC, "Serial Number")
    field(SCAN, "I/O Intr")
    field(DTYP, "asynInt32")
    field(INP, "@asyn($(PORT),42)UINT16")
}
```

ADS module

Device Address = Device Function + Arguments

```
record(longin, "$(P):sym_int32_counter") {
    field(DESC, "Read ADS by symbolic name")
    field(DTYP, "asynInt32")
    field(INP,  "@asyn($(PORT)) DINT R P=$(ADS_PORT) V=TestPlan.sym_int32_counter")
    field(SCAN, "I/O Intr")
}
```

Read and write handlers in autoparamDriver

```
Result writeWord(DeviceVariable &var, epicsInt32 value);

Result<epicsInt32> readWord(DeviceVariable &var);

void MyDriver::MyDriver(...) {
    ...
    registerHandlers<epicsInt32>("WORD", readWord, writeWord, NULL);
    ...
}

record(longin, "$(P):sym_int32_counter") {
    field(DTYP, "asynInt32")
    field(INP,  "@asyn($(PORT)) WORD 0x1234")
}
```


EPICS COSYLAB

Addresses and variables

```
class DeviceAddress {
    virtual bool operator==(DeviceAddress const &other) const = 0;
};

class DeviceVariable {
    std::string const &function() const;
    DeviceAddress const &address() const;
    int asynIndex() const;
    asynParamType asynType() const;
};
```

These are meant to be subclassed.

EPICS COSYLAB

Interrupts

Interrupts can be processed:

- during or after running write or read handlers,
- in response to hardware interrupts (e.g. from callbacks),
- at any other time, e.g. from a background scanning thread.

Device variables are backed by asyn parameters

```
class Driver : public asynPortDriver {
protected:
    template <typename T>
    asynStatus setParam(DeviceVariable const &var, T value,
                        asynStatus status = asynSuccess,
                        int alarmStatus = epicsAlarmNone,
                        int alarmSeverity = epicsSevNone);

    template <typename T>
    asynStatus doCallbacksArray(DeviceVariable const &var, Array<T> &value,
                                asynStatus status = asynSuccess,
                                int alarmStatus = epicsAlarmNone,
                                int alarmSeverity = epicsSevNone);
};
```


EPICS COSYLAB

Learning about interrupt subscriptions periodically

```
class Driver : public asynPortDriver {
protected:
    std::vector<DeviceVariable *> getAllVariables();

    std::vector<DeviceVariable *> getInterruptVariables();
};
```

Learning about interrupt subscriptions immediately

```
asynStatus intrRegistrar(DeviceVariable &var, bool cancel);

void MyDriver::MyDriver(...) {
    ...
    registerHandlers("WORD", readWord, writeWord, intrRegistrar);
    ...
}
```

https://epics.cosylab.com

Thank you