ITER I&C CODAC

55.C4 Divertor Thomson Scattering

Slow Controller Example

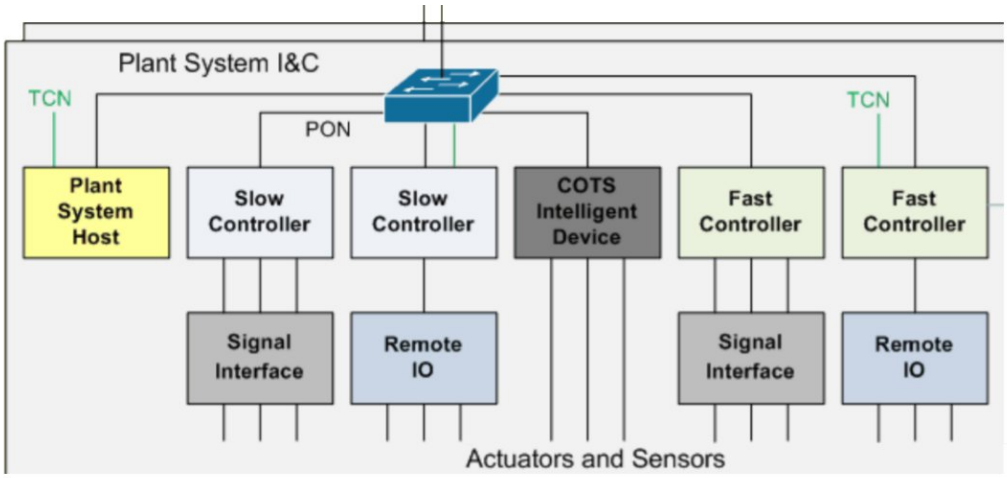
Ioffe inst. Russia, Igor Bocharov

Данный документ кратко описывает основы использования медленных контроллеров в составе системы управления токамаком I&C CODAC проекта ITER. Документ не заменяет основные документы по подсистеме, а помогает понять что они там написали и имели ввиду. Очень кратко. Выжимка. Простым языком.
Возможны ошибки, написано по памяти.

Оглавление

Общее

Для управления диагностикой из машинного зала, а также для удалённого доступа экспериментатора к диагностике, необходимо, что бы диагностика имела интерфейс к системе ITER I&C CODAC PBS 45. Для этого диагностика должна содержать обычный компьютер архитектуры x86 с установленной ОС RedHat, на него ставится ОС для проведения ядерных экспериментов EPICS, на него ставится инструмент для конфигурирования баз данных EPICS (SDD-Editor) и визуализации переменных EPICS (CS-Studio). Этот компьютер подключён во внутреннюю сеть ITER и обменивается с ней переменными EPICS (EPICS PV - process value, про Epics придётся почитать), таким образом и осуществляется обмен данными и управление диагностикой. Архитектура I&C CODAC подключения диагностики указана на картинке ниже.



Тут есть методологическая ошибка, важная для понимания Slow Controller, которая копируется с DDD в остальные документы: медленный контроллер не подключается в PON сеть напрямую для обмена переменными EPICS, а он соединяется с быстрым контроллером, а тот уже, через Plan_System_Host, конвертирует обмен Slow в обмен с Fast и далее. Также не в том месте отрисован PSH - это софт. компонент, это может быть виртуальная машина на Fast, или просто набор процессов EPICS; планируется мигрировать PSH в виртуальные машины внутри датацентра ITER.

Если в вашей системе планируется управление медленными процессами, которым не нужна точная синхронизация с временем токамака, и если эти процессы управляются отдельными проводами с дискретными или аналоговыми сигналами, то возможно в вашу систему имеет смысл поставить медленный контроллер (Slow Controller), который представляет собой обычный промышленный программируемый логический контроллер (PLC).

Ниже в документе в общих чертах рассказывается как написать приложение на Slow, что бы оно успешно работало с CODAC и Fast контроллером. Ссылка на исходные коды:
github.com/gans-spb/55.C4-SlowConTest

Состав оборудования

Для работы диагностики вам нужен:

- Fast Controller - ПК который умеет обмениваться PV EPICS, требуется что бы он был на RedHat
- Slow Controller - ПЛК, который имеет драйвер к Fast, и уже через Fast осуществляется обмен.

> Важно! **Различия Fast и Soft**:
>
> **Fast** используется для регистрации событий с высокой разрешающей способности по времени и с высокой точностью, т.к. он имеет малый квант времени и привязку к сети TCN протокола точного времени PTP IEEE 1588 (или через плату времени NI, или возможно через сетевой адаптер i350 но будет больше джиттер)
>
> **Slow** используется там, где неважна привязка времени, т.к. у slow низкая точность своего RTC и редкий обмен с Fast, и где есть прямое управление отдельными электрическими сигналами.

Slow controller

Это просто покупной ПЛК. В требованиях ИТЕР или Siemens Simatic или Schneider. Т.к. Simatic занимает львиную долю рынка автоматизации, то далее про него. На настоящий момент времени идёт генерация s7, сейчас (2022) в ней серия 300 снята с закупок (EOL), то смотрим серию 1200 и 1500. 1200 дешёвый, поэтому его ставят только в мониторинг параметров стойки (Cubicle Monitoring), поэтому надо брать серию 1500. В случаях особой надёжности берём серию с резервированием, для неё же свой драйвер.

Т.к. за время создания ИТЕР даже «вечный» Siemens успел сменить линейку оборудования, то разобраться в кипах документации ИТЕР тяжело (в конце списки полезных документов), а в коде индусских товарищей ещё сложнее, поэтому далее кратко выжимка.

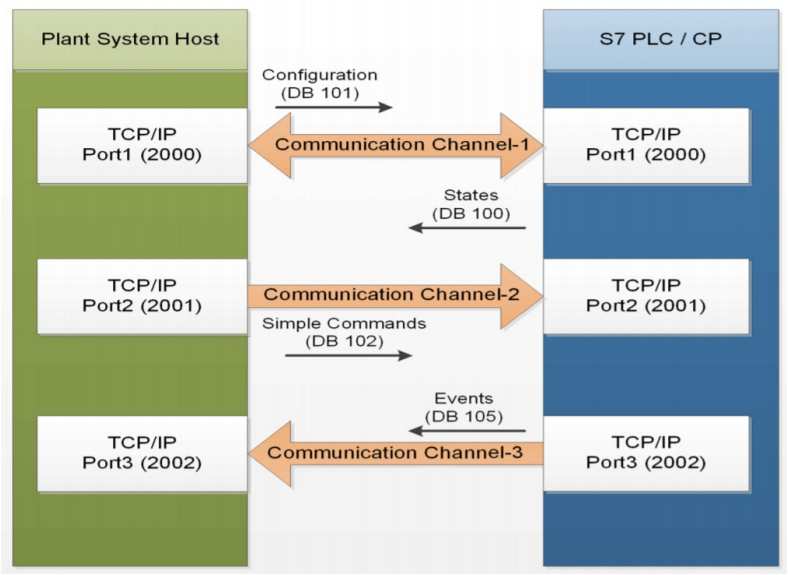
Подключение Slow-Fast

- Slow подключается к центральной системе через Fast, т.е. они должны быть в одной Ethernet сети.
- Для этого у Fast должен быть драйвер Epics, он называется s7asyn
- На Fast запускается экземпляр Epics (IOC) в котором в его DB прописаны переменные Epics
  Epics после этого ожидает обмена по этим переменными со Slow
- Обмен идёт по TCP/IP, порты прописаны в конфиге IOC
- Slow ничего не знает про ITER, это покупное изделие из магазина.
  Slow надо иметь драйвер с CODAC, он называется SPSS
  Spss надо собрать в Tia/Step и закачать в Slow
- Таже ручками в DB объекты PLC надо прописать переменные, которыми будет обмен
- После этого при старте Slow откроет три порта по которым пойдёт обмен с IOC EPICS
- Bingo!

вышеуказанный процесс легко воспроизвести обычному около-ИТ товарищу буквально за пол года.

Протокол обмена Fast-Slow

Конкретизируем концепцию обмена. Есть *четыре* канала обмена Fast-Slow, первые два которых посажены на один IP порт, остальным по отдельному порту. Slow открывает три IP порта как сервер и ждёт обмена по ним, см картинку. Третий порт появился позже всех, поэтому в CubicleMon его нет.



States - это переменные состояния PLC, высылаются на регулярной основе, как ответ на Config от Fast

Config - это переменные конфигурирования PLC, высылаются на регулярной основе от Fast

Command - это одиночные биты управления PLC, , высылаются на регулярной основе от Fast

Event - это *срочная* очередь возврата от Slow в Fast битовых значений какого-то процесса

Со стороны Fast за обмен отвечает экземпляр Epics IOC, в котором сконфигурированы порты, и в DB которого указаны переменные для обмена. Со стороны Slow за обмен отвечает программный код SPSS, который понимает формат пакетов и данных от Fast. Что бы было сложнее, Fast анализирует заголовок States и формирует данные состояние обмена (sys_mon). Внутри Slow логику обработки переменных и возврата значений пишет программист, исходя из логики работы диагностики. Сам протокол примитивный, данные пишутся в сыром виде напрямую в пакет TCP.

Каналы обмена

Размер пакетов данных для портов 2000 и 2001 жёстко задан в базе [DB104], они должны быть строго синхронны размерам в конфигурации IOC у Fast. Канал Command работает только на приём, поэтому send_len=0. Event канал сделан совершено по другому.

Name	Data type	Offset	Start value
Static			
channel1	"UCodacChannel"	0.0	
SEND_LEN	Int	0.0	64
RECV_LEN	Int	2.0	2
channel2	"UCodacChannel"	4.0	
SEND_LEN	Int	4.0	0
RECV_LEN	Int	6.0	4

States.

Структуры данных находятся в Slow в [DB100]. Направление от Slow к Fast. 10ms.
Видно, что исторически это первый канал обмена. Пакет данных содержит фиксированный по структуре и размеру Header, в котором мы видим, последовательно:

- магическое слово, оно же в конце
- общая длина пакета, с Header и Footer, такая же должна стоять в [DB104]
- строка версии, аж 40 символов (привет «экономии» трафика) строго соответствует версии в IOC в Fast
- таймстемп, ставит Slow исходя из своего RTC, не более чем на 5 минут сдвиг от времени Fast, причём время у этих некрофилов в BCD
- далее идут последовательно побайтно данные
- в конце - Footer, который просто магическое слово

Name	Data type	Offset	Start value
Static			
Header	"CodacStatesHe...	0.0	
FixedPattern	DInt	0.0	16#2F08000
Length	Int	4.0	64
InterfaceVersion	String[40]	6.0	'Cub_Mon_Proto'
AliveCounter	Int	48.0	0
TimeStamp	"utDate_And_Time"	50.0	
Year	Byte	50.0	16#0
Month	Byte	51.0	16#0
Day	Byte	52.0	16#0
Hour	Byte	53.0	16#0
Minute	Byte	54.0	16#0
Second	Byte	55.0	16#0
Msec1	Byte	56.0	16#0
Msec2wDay	Byte	57.0	16#0
ao16	Int	58.0	123
Footer	"CodacStatesFooter"	60.0	
FixedPattern	DInt	60.0	DINT#4245651...

Такой пакет Slow автоматически начинает слать 10 раз в секунду после поднятия соединения. Все проблемы соединения IOC автоматически отображает в мониторинге как пропущенные пакеты и.т.п. Оно же отражается в логах. Любое отклонение пакета от нормы и от частоты посылок и от времени timestamp - Fast сразу рвёт соединение, всё останавливается - вы даже не заметите, вот такая «надёжность».

Configuration

Структуры данных находятся в Slow в [DB101]. Направление от Fast к Slow. 10ms. Просто в лоб поток бинарных данных. Естественно, он должен совпадать в Epics и Slow до бита, или будет вылет за память. Считается, что этими данным мы конфигурируем систему перед стартом. Но т.к. после старта нет других каналов обмена переменными, то это единственный канал что-то стоящее сообщить в Slow. Судя по логике - нужен для установки значения одного аналогового выхода.

Name	Data type	Offset	Start value
Static			
ai16	Int	0.0	1

Commands

Структуры данных находятся в Slow в [DB102]. Направление от Fast к Slow. 10ms.

Просто пакет однобитовых флажков от контроллера к Fast. Реализована логика “Spring-button”, когда флажок сбрасывается и Fast после передачи и Slow после приёма. Можно сделать Byte передачу, но передастся всё равно один бит. Судя по логике - нужен для активации одного цифрового выхода.

Name	Data type	Offset	Start value
Static			
CMD1	Byte	0.0	16#0
CMD2	Byte	1.0	16#0
CMD3	Byte	2.0	16#0
CMD4	Byte	3.0	16#0

Events

Структуры данных находятся в Slow в [DB105]. Направление от Slow к Fast. ASAP.

Крайне странная идея отсылки опять таки битовых флажков от Slow. Сделан хитро. Особенности: передаётся сразу, наличие очереди передачи, тамймстеп для каждого сообщения очереди отдельно. По описанию - нужен для отслеживания событий *более частых*, чем частота обмена с Fast, например для какого-то current trip в течении нескольких ms. При этом Slow может накопить эти события, отметить метками времени, и далее кучей передать. На стороне Fast задача программиста как обработать логику этих событий.

Структура подобна States, но с счётчиком фреймов. Высланный фрейм на стороне Slow затирается. Пакет высылается сразу, как появляется, без таймаута.

Name	Data type	Offset	Start value
Static			
Header	"_CodacEventsH..	0.0	
FixedPattern	DInt	0.0	DINT#49315840
TimeStamp	"utDate_And_Time"	4.0	
FrameCounter	Int	12.0	0
StatusBits	Array[0..15] of Bool	14.0	
StatusBits[0]	Bool	14.0	FALSE
StatusBits[1]	Bool	14.1	false
StatusBits[2]	Bool	14.2	false
StatusBits[3]	Bool	14.3	false
StatusBits[4]	Bool	14.4	false
StatusBits[5]	Bool	14.5	false
StatusBits[6]	Bool	14.6	false
StatusBits[7]	Bool	14.7	false
StatusBits[8]	Bool	15.0	false
StatusBits[9]	Bool	15.1	false
StatusBits[10]	Bool	15.2	false
StatusBits[11]	Bool	15.3	false
StatusBits[12]	Bool	15.4	false
StatusBits[13]	Bool	15.5	false
StatusBits[14]	Bool	15.6	false
StatusBits[15]	Bool	15.7	false
Length	Int	16.0	0
Reserved	Word	18.0	16#0
Footer	"CodacStatesFooter"	20.0	
FixedPattern	DInt	20.0	16#FD0F7FFF

Структура программных блоков TIA

Структура блоков особо не нужна, но для общего развития вот очень приблизительно она, из одного из многих неправильных документов ITER.

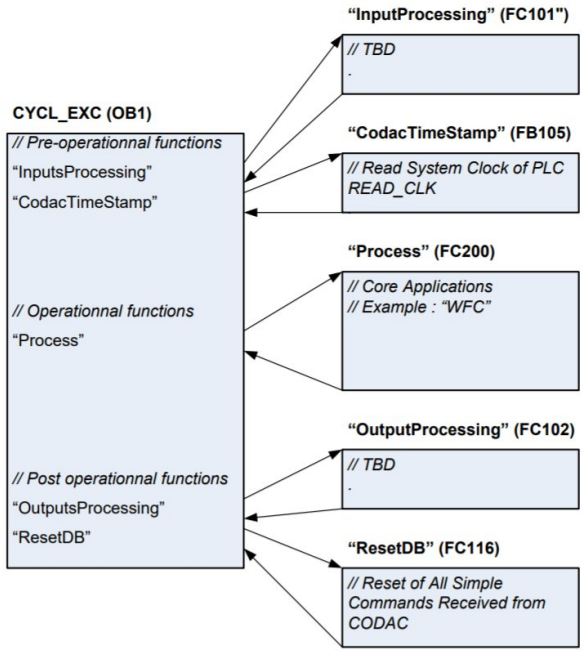


Figure 15 : Main Cycle Loop Standard Structure

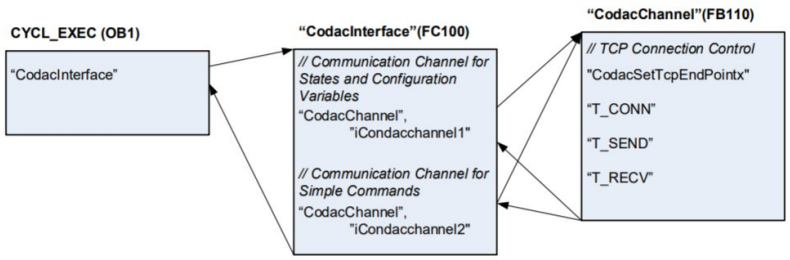


Figure 16 : Main Cycle Loop Standard Structure

Сначала инициализируются размеры пакеты данных и номера портов.

Далее поднимаются сокеты и переходят на приём:

CodacCommInterfacePlc -"iStateConfigChannelPlc" для Config и State одновременно,

CodacCommInterfacePlc - "iCommandChannelPlc" для Command отдельно,

и в конце "EventInterface".

В каждом из них происходит обновление счётчика кадров, таймстемп, и прочее.

Необходимо отметить, что в разное время SPSS писали разные индусы на разном оборудовании, поэтому сложно описать весь огород и всю эволюцию версий SPSS и его место в современной разработке. По уму я бы всё переписал, будет короче и понятнее.

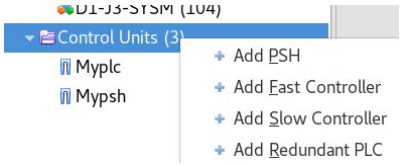
Создание проекта

Часть Hardware

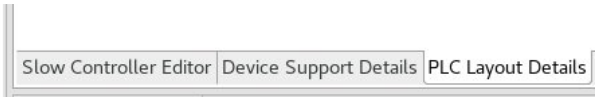
1. Берём ПК, можно виртуалку, ставим CODAC (у меня 6.3)
2. Берём Simatic, у меня был s7-1200, и я портировал код 1500 на 1200, это была боль
3. Включаем всё в один сегмент сети, что бы Simatic точно пинговался с Fast.

Часть CODAC

1. Создаём новый проект в SDD
2. Важно - все названия всего - вообще не важны! EPICS воспринимает любые строки, хоть просто строка «genby–666». Но лучше в нотации ITER, например «D1-J3-xxx-yyy». Названия PLC и PSH тоже любые, также как и их версии и типы в настройках.
3. Добавляем PLC и PSH в Control Units.



4. Нижние закладки в настройках PLC. В Device_Support_Details сразу встанет s7asyn. В PLC_Layout_Details можно ничего не трогать, но посмотреть как он создал по сути три порта и номера баз данных. Галочку «Bool/Byte» для Command можете не трогать, т.к. индус уже угрохал 40 байт на номер версии в Config.

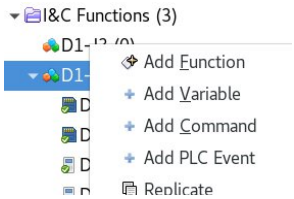


5. У PLC ставим в PON адрес PLC (Всё остальное можно localhost)

Network	Name
CIN	
PON	172.16.19.220

6. PLC присваиваем PSH как хост для IOC

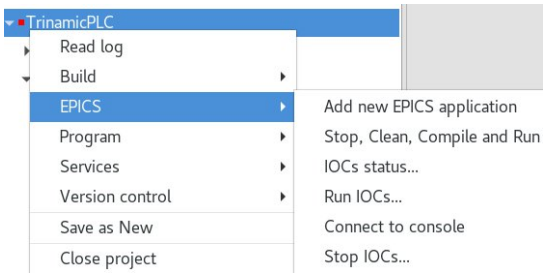
7. Добавляем переменные, какие хотим, под какую-либо функцию. Command и Event добавляются отдельно.



8. В DeploymentTarget переменных ставим наш PLC (сразу увидите IOCname, который создастся).

9. Дальше генерируем конфиг файлы и переходим на дерево «Maven Editor»

10. Тут интересен компайл и запуск Ioc, и всё.

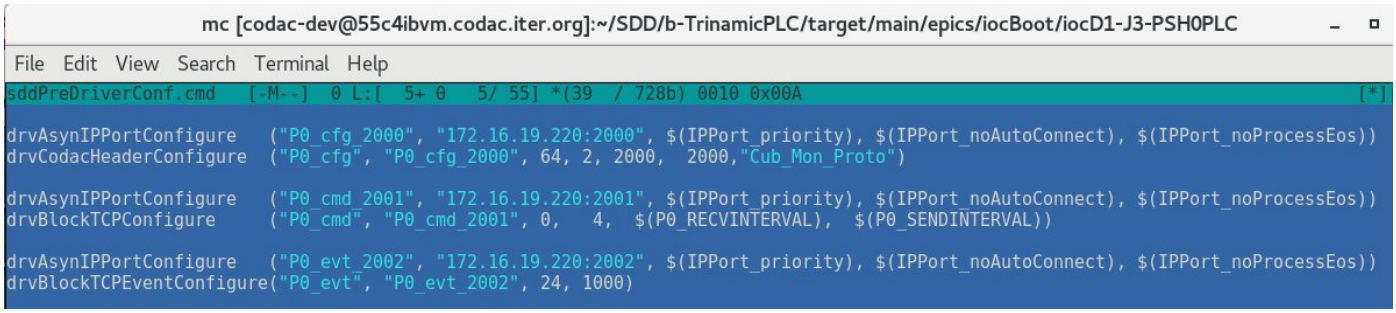


11. Так, далее открываем капот и лезем в кишки. Запускаем консоль, например mc, и лезем в исходники. Нам интересен ~/SDD/b-PrjName/Src. С него он билдит Target, творческим копированием.

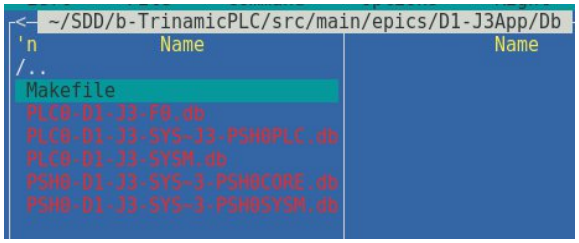
12. Fast собирает себе «экзешник» - запускаемый экземпляр Epics, он называется IOC, с базами данных переменных, какие вы указали в SDD. Ниже - код самого IOC, там ничего нет, всё управляется через DB.

```
int main(int argc,char *argv[])
{
    sigset(SIGTERM, epicsExit);
    if(argc>=2) {....
        iocsh(argv[1]);
        epicsThreadSleep(.2);
    }
    iocsh(NULL);
    epicsExit(0);
    return(0);
}
```

13. В epics/iocBoot лежат скрипты старта IOC. Там страшная красноглазая свалка, нам же нужен только PLC и только sddPreDriverConfig. Можно сразу в target, т.к. каждый compile он будет брать из src и перезаписывать target. Там три магических строки конфигурации портов s7asyn данного ioc. Читайте PJAHXJ.
drvAsynIPPortConfigure - подъём порта
- ("P0_cfg_2000", "172.16.19.220:2000" - тег порта, реальный ip и порт PLC(!),
- $(IPPort_priority), $(IPPort_noAutoConnect), $(IPPort_noProcessEos)) - мелочи
drvCodacHeaderConfigure - указываем как парсить заголовок
-("P0_cfg", "P0_cfg_2000" - связываем тег порта и тег в конфиг файлах ioc - будет ниже
- 64, 2, - in/out размер пакета, в байтах, с хеадером, должно совпадать с размером баз в PLC [DB104]
- 2000, 2000, - таймауты приём и передача, 2s для простоты отладки WaveShark
- "Cub_Mon_Proto") - строка версии протокола, должна совпадать со строкой в PLC



14. DB - это базы данных Epics для старта IOC, там все переменные, их типы, куча всего. Тут DB под PLC, PSH, Sys monotor и.т.п. Смотрим DB которая наша функция типа PLC-D1-J3-F0.db



15. Структура DB такая. Каждая переменная - одна запись. Там тип Epics (ai, ao,...), тип Asyn (DTYP, “asynInt”), и важнейшее поле (INP, "@asyn(P0_cfg 58)INT16"):
- INP/OUT - парсить входной пакет или формировать исходящий
- @asyn - маркер драйвера Asyn, обработка PV будет через драйвер
- P0_cfg - через какой порт слать или получать, см. конфиг портов
- 58 - это указатель (pointer) сдвига от начала пакета, включая заголовок для State, вот он корень зла

- INT16 - сколько битов под параметру отвести откуда s7asyn драйвер будет выдирать/вставлятьпеременную. Очевидно, что смещение должно совпадать с началом соотв. байта в пакете ip. Вот тут у меня ни один SSD проект не сошёлся по смещениям и я ставил руками.
Остальные параметры некритичны для обмена с PLC.

```
/home/codac-dev/SDD/b-TrinamicPLC/src/main/epics/D1-J3App/D
record (ai,"D1-J3-F0:Var16in")
{
        field(DTYP, "asynInt32")
        field(INP, "@asyn(P0_cfg 58)INT16")
        field(PINI, "YES")
        field(SCAN, "I/O Intr")
        field(SIML, "D1-J3-SYSM-:PLCMyplc-SIM-NOPLC CP")
        field(SSCN, ".5 second")
        field(TSE, "-2")
}
```

16. Неожиданно важнейшая структура данных лежит в PSH0PLC.db, оказывается драйвер там конфигурирует переменные Sys Monitoring, без которых не может жить. В нём маппинг загадочных sys_mon параметров на смещения в заголовке State пакета PLC, по которым драйвер понимает, что обмен нормальный. Ошмётки этих данных можно увидеть в документах по SysMon (не путать с Cubicle Monitor). Я до конца не понял, какие из них важны, а какие нет, но без них валится.

```
/home/codac-dev/SDD/b-TrinamicPLC/src/main/epi~1-J3App/Db/P
record (ai,"D1-J3-SYSM-:PLCMyplc-ALIVEC")
{
        field(DESC, "PLC alive counter")
        field(DTYP, "asynInt32")
        field(INP, "@asyn(P0_cfg 60)UINT8")
        field(PINI, "YES")
        field(SCAN, "I/O Intr")
        field(SIML, "D1-J3-SYSM-:PLCMyplc-SIM-NOPLC CP")
        field(SSCN, ".5 second")
        field(TSE, "-2")
}
```

17. Также можно посмотреть, как SDD сгенерировал структуры данны для PLC для TIA/Step.

```
~/SDD/b-TrinamicPLC/src/main/plc/Myplc/sdd
'n          Name                    Name
/..
 Myplc-CodacInterface.awl
 Myplc-CodacInterface.sdf
 Myplc-CodacInterface.xlsx
```

18. Когда у нас настроен Ioc pre .cmd и правильно заданы смещения в DB, можно запускать экземпляры Ioc, их стартует три штуки.

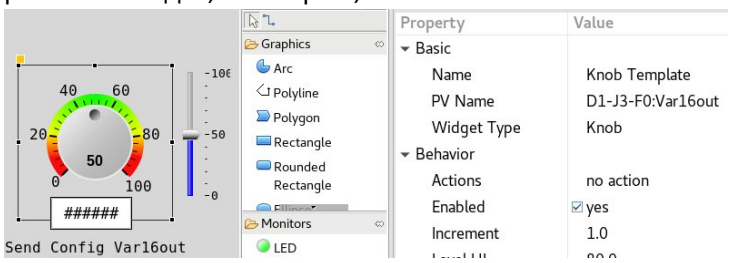
```
<Closed> Maven terminal
[INFO] Start iocD1-J3-PSH0SYSM on localhost
Starting project IOC 'iocD1-J3-PSH0SYSM'...          [  OK  ]
[INFO] Start iocD1-J3-PSH0PLC on localhost
Starting project IOC 'iocD1-J3-PSH0PLC'...           [  OK  ]
[INFO] Start iocD1-J3-PSH0CORE on localhost
Starting project IOC 'iocD1-J3-PSH0CORE'...          [  OK  ]
```

19. После запуска Ioc сразу начинает поддерживать свои переменные и ими можно управлять через caget и caput.

```
File  Edit  View  Search  Terminal  Help
[codac-dev@55c4ibvm ~]$ caget D1-J3-F0:Var16in
D1-J3-F0:Var16in               18
[codac-dev@55c4ibvm ~]$
```

20. Что бы не уродоваться в консоли, в CS-Studio можно накидать простой OPI интерфейс, вытащив наружу переменные, которые вы ввели в DB. Если элемент розовый - значит OPI не видит переменной. Если не

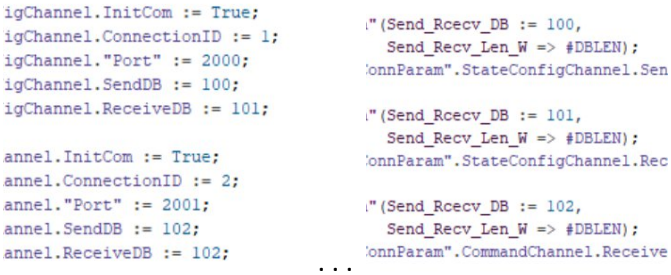
розовый - видит, но не факт, что есть коннект к Slow. Поэтому переходим к Slow.



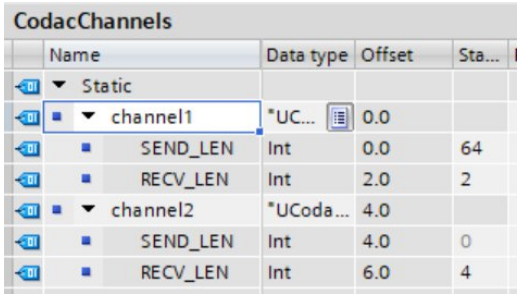
Часть Simatic

1. Ставим TIA Portal (v17), значит все взятые SPSS, будут конвертированы в него, и не всегда успешно.
2. Скачиваем любой SPSS под свой контроллер (та ещё задача), и запускаем проект в TIA
3. Проект сразу должен собраться и загрузиться. Если нет - печаль.
4. В проекте должны появиться базы для Config, State, Command и Channels. Возможно и для Events.

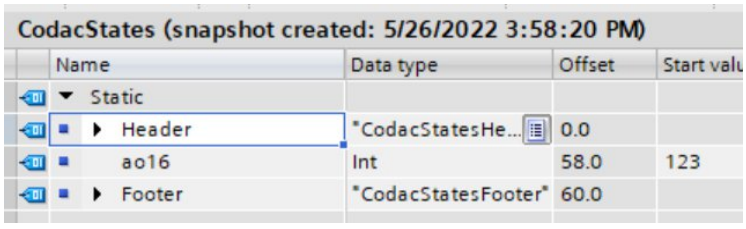
5. Номера баз у индусов жёстко прописаны в коде, все-таки термояд, поэтому если они при импорте сбились - лучше вернуть их номера.



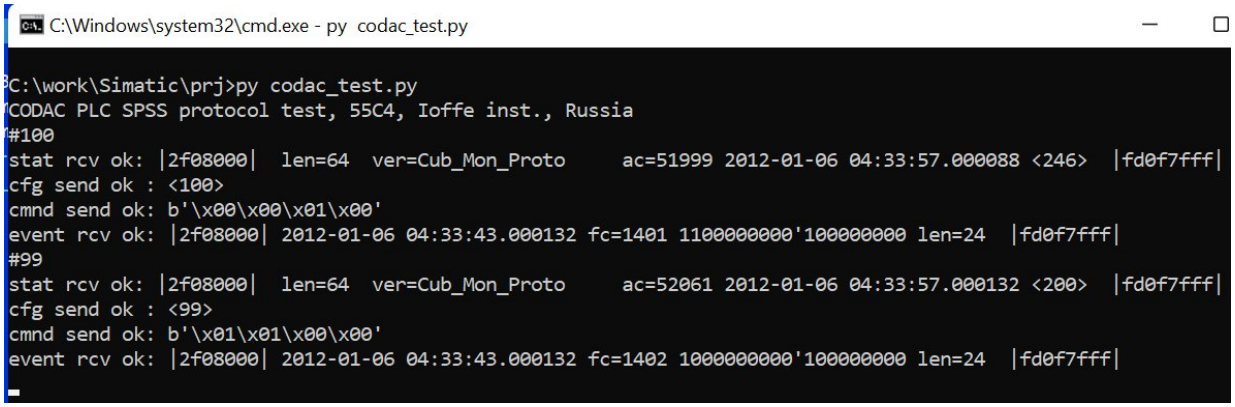
6. Уточняем также размеры приём/передача в каналах, должно строго соответствовать конфигу соответствующего порта. Command канал на передачу всегда ноль. Event живёт отдельной жизнью.



7. Все пакеты States, Config, Cmd, должны строго соответствовать тому, что вы в конфигах DB написали. Например у меня ниже одна переменная 16 бит. Теоретически надо взять awl/sdf файл, сгенерированный SDD, и импортировать в TIA, и все данные автоматически встанут на свои места, но у меня так не произошло. Поэтому лучше глазами проверить, что все смещения в байтах ровно такие, какие указаны в DB ioc.



8. Компилим-заливаем-стартуем. Если всё ок, то в PLC открываются порты 2000-2001-2002, Ioc к ним коннектится, и начинается обмен. Его можно перехватить WaveShark, там всё просто. Если всё сложно, то можно остановить Ioc, и использовать мой py скрипт по «реверс ижинирингу» протокола codac_spss_test.py. Он коннектится ко всем трём портам и изображает обмен.



Итого, повторяем последовательность действий

1. В SDD создаём проект, в котором участвует PLC. PLC связывается с PSH. Введённые переменные, ивенты и команды связываются с PLC. Генерируем проект, проверяем в консоли что там нагенерилось, наверное правим DB и конфиги pre .cmd. Компилируем и запускаем IOC.

2. В TIA импортируем проект SPSS. Уточняем что номера DB корректные, типа DB10x. Импортируем переменные из SDD проекта в виде awl/sdf файлов, или вручную вставляем в DB блоки. Пишем какую-то логику, или не пишем. Стартуем PLC.

3. В CS-Studio рисуем OPI интерфейс на основе использованных переменных, запускаем его, смотрим изменение переменных, радуемся.

CubeMon

Программу Cubicle Monitoring писал совсем другой индус, но она может быть полезна для обучения. Она очень плохо написана макаронным кодом. Есть много доков именно по CubeMon. Зато этот проект должен работать сразу, на серии s7-1200.

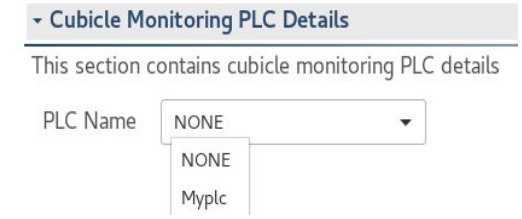
Технически CubeMon это PLC s7-1200 AC (без БП, сразу в розетку, может иметь фазу на клеммнике), у него есть сигналы на вход: «дверка открыта перёд», «дверка открыта зад», «автомат вентилятора отщёлкнулся», «температура»; на выход лампочка «Fault» (дверка открыта, вентиль умер или температура выросла за порог). На планке специально максимально запутано куча колодок и автоматов, а также дорогой плохой термометр через одно место. Исходя из кода, один PLC с 16DI/DO может обслужить 11 стоек.

Кратко надо так:

1. TIA, ищем на ITER и загружаем проект типа plc-1BG40, там куча бойлерплейт кода. Суть простая - по количеству кубиклов в Fast высылаются такие структуры: две дверки, карлсон, fault, температура. Со стороны Fast ничего не идёт.

TEST_S7_COMM_CUB01	"sTEST_S7_COMM"
CBS1-CBS2-SYS:HTH-CU-CY1	Bool
CBS1-CBS2-SYS:HTH-CU-CY2	Bool
CBS1-CBS2-SYS:HTH-CU-FAN	Bool
CBS1-CBS2-SYS:HTH-CU-Spar...	Bool
CBS1-CBS2-SYS:HTH-CU-TT1	Real

2. В SDD надо завести PLC как мониторинг, а не как объект, к которому потом привязываются переменные. Это делается через добавление кубикла, и уже к нему вяжется PLC, который становится мониторингом. Под него автоматом генерятся переменные sys_mon и базовый интерфейс мониторинга opi.



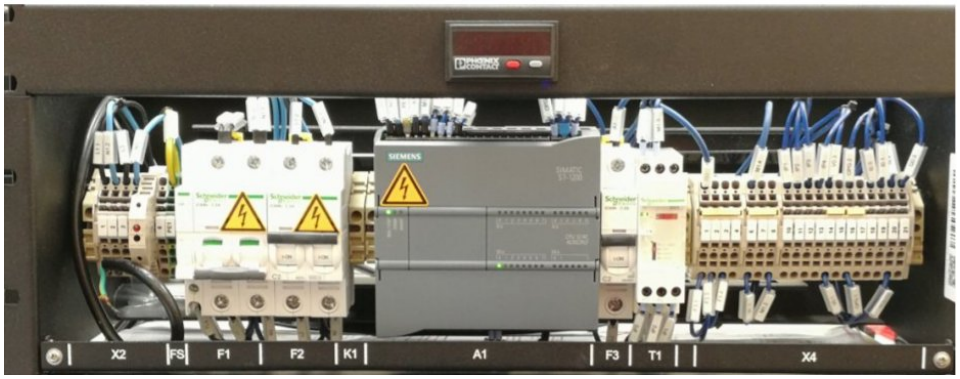
3. После генерации SDD проекта, в CS-Studio закладках мониторинга в OPI, появятся картинки для CubeMon.

Name	Physical Name	Description	CUBHLTS	PLCHLTS
C0	410000-CU-0001	first cubicle	Ok	Communication with PLC OK
C1	410000-CU-0002	second cubicle	Door(s) open n	Communication with PLC OK
C2	410000-CU-0003	third cubicle	Door(s),FAN,Tem	Communication with PLC OK
C3	410000-CU-0004	fourth cubicle	Door(s),FAN,Tem	Communication with PLC OK
C4	410000-CU-0005	fifth cubicle	Door(s),FAN,Tem	Communication with PLC OK

4. После запуска всего этого хозяйства переменные должны быть видны (не розовые) и замыкая нужные контакты клеммника можно имитировать открывание дверок и Fault.

Моё мнение, что CubeMon надо показывать как не надо делать:
он плох как со стороны железа, уныл как софт, и архаичен как проект.

Фото CubeMon для идентификации железа:



Прототип для диагностики 55.C4

Состав стенда

- **Simatic s7-1200** с платой CB1241 для RS485 и модулем 16 DI/DO
  программируется в Simatic TIA Portal v17
- **Simatic Logo!** для симуляции управляющих сигналов на s7-1200
  программируется в Logo Soft Comfort
- **Trinamic TMCM-3110** как контроллер шаговых двигателей
  управляется через TMCL-IDE
- **Moxa NPort 5130** для проверки управления Trinamic напрямую из Fast без Slow (опционально)
  управляется через MXview
- **STM32F746G-disco** плата STM32 для имитации по RS485 диагностического лазера (опционально)
  программируется в CubeIDE и TouchGFX

Фото стенда:

Исходники

plc-Trinamic485 - Slow часть для Simatic s7-1200 для обмена с CODAC Fast

plc-1BG40-3.0_V17 - Slow часть для Simatic s7-1200 для кубикл-мониторинга

plc-LogoTest - Slow часть для Simatic Logo! для тестирования s7-1200

b-TrinamicPLC - Fast часть, проект SDD содержащий настройки Epics IOC и GUI для CS Studio

STM32-laser-modbus - симулятор оборудования (лазера) под RS485 шине, для отладки обмена

Trinamic485

Проект управления контроллером шаговых двигателей Trinamic, основной проект.
Логика расписана в .md, тут кратко:

- Порт SPSS s7-1500 на s7-1200
- Обмен данными с Fast по IP со структурами s7asyn, все четыре канала, включая Event
- Управление двигателем: старт по Cmd1, скорость от Var16out, стоп по таймауту
- Ворврат в Fast значения Var16in = 2 * Var16out, высылка Event1 и Event2
- Мигание лампочками, управление двигателем от Input от PLC Logo

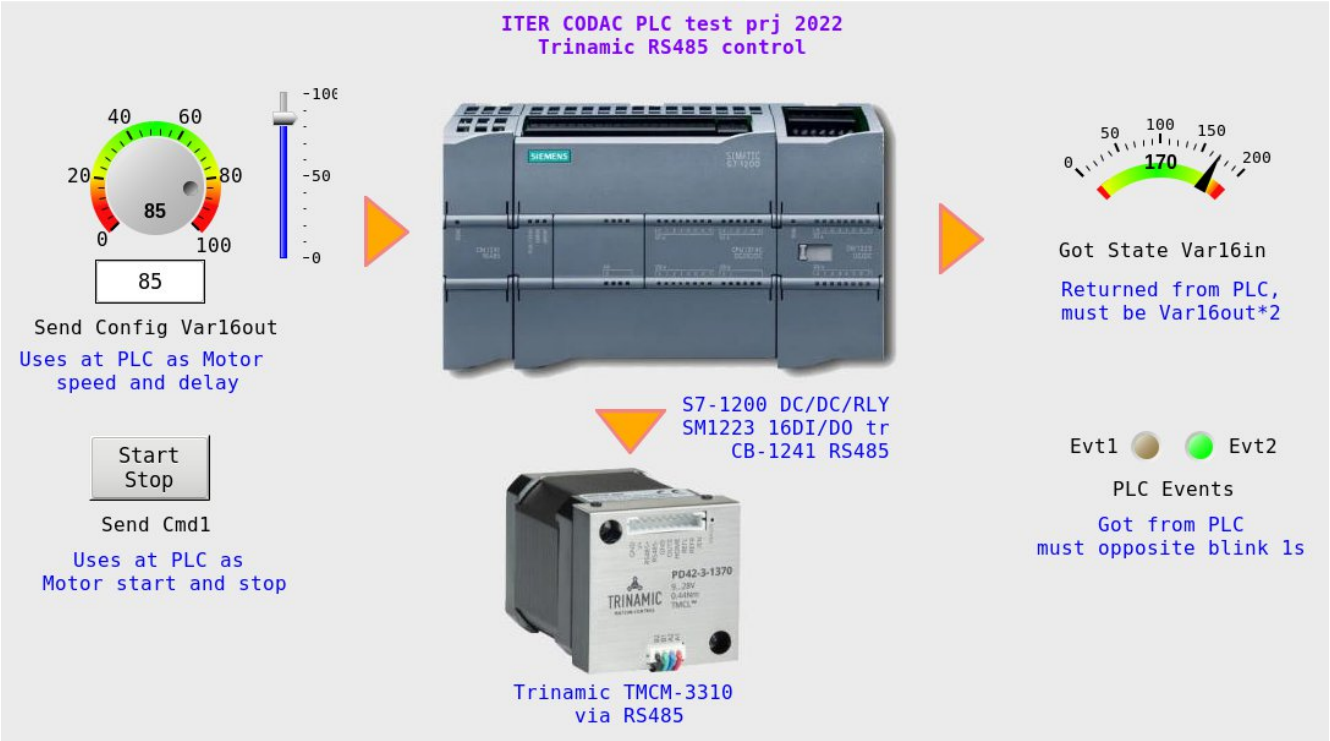
Интерфейсы прототипа

OPI интерфейсы прототипа управления медленным контроллером. Запускаются в CS-Studio на Fast и через IOC, запущенный в SDD, работает в паре с Slow.

SDD Проект b-TrinamicPLC, требует:

- CODAC 6.3
- s7asyn, в составе CODAC
- SDD 6.3.1, в составе CODAC
- CS Studio 4.7, в составе CODAC

Создание проекта, сборка и запуск много где описано у ИТЕР. Особенности с PLC изложены выше в «Часть Simatic». Интерфейс лежит в SDD/full_hd/Trinamic485.opi



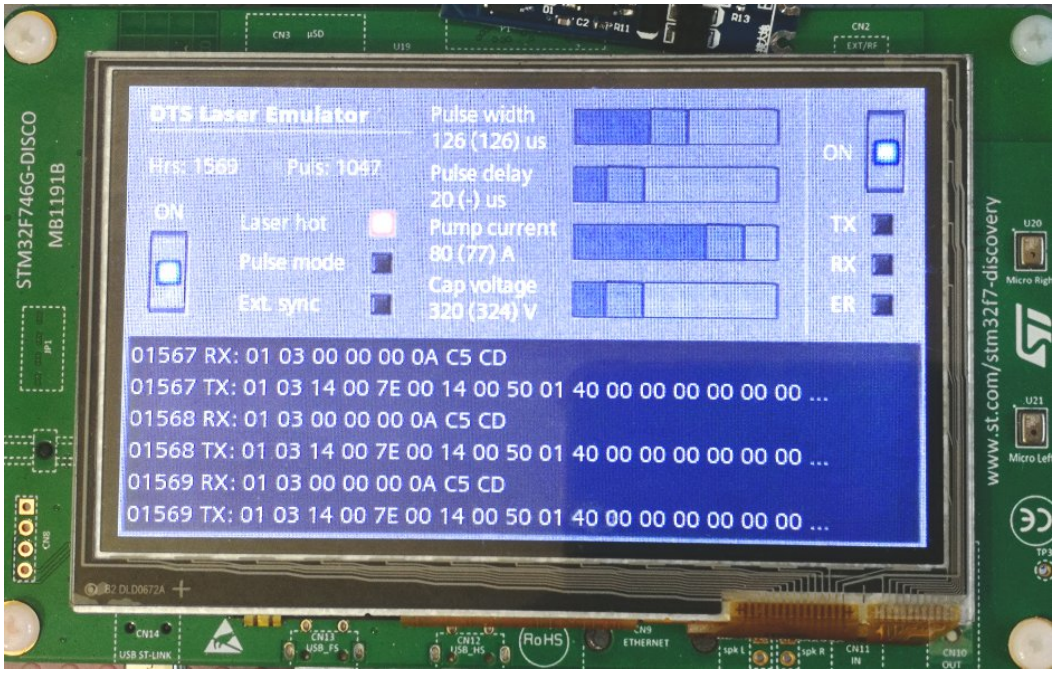
Логика работы:

- Переменная Var16out, установленная слева, должна пройти через PLC и вернуться ровно удвоенной в Var16in.
- Она же (вместе с ADC1) устанавливает скорость двигателя и время вращения (обратно скорости).
- Команда “Cmd1” стартует мотор, стоп по таймауту
- События Evt1 и Evt2 шлются со стороны PLC с частотой 1/2 Hz поочерёдно.

Если Slow запрограммирован верно, и виден в сети, и IOC запущены, то всё должно работать.

Симулятор лазера

Простое ПО под дев-борду STM32 для симуляции приборов, работающих по RS485 - если прибора ещё нет, а обмен с ним уже надо программировать.

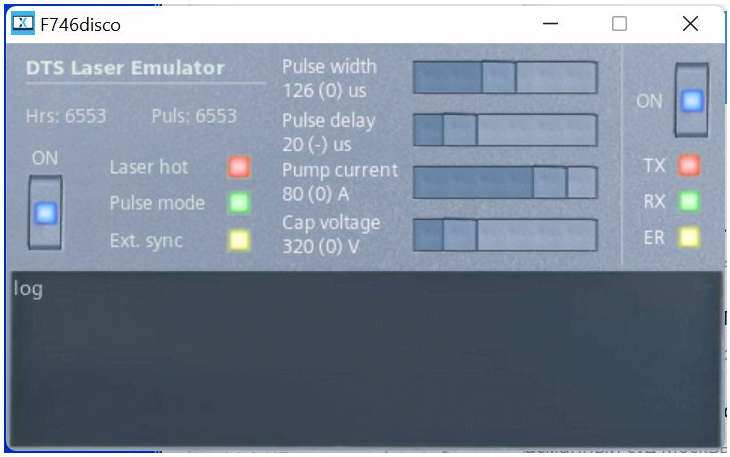


Проект собран под F7 с индикатором, но если выкинуть GUI, то можно портировать на любой STM32. Умеет производить обмен по RS485 и симулировать работу диагностического лазера. Имеет удобный интерфейс и журналирование обмена. Можно доработать до симуляции любого реального прибора для целей тестирования, или использовать как сниффер RS485 протокола.

Требует:

- 32F746G-DISCOVERY = STM32F746NG + LCD 4.3” 480×272 FCMS + SDRAM
- RS485 берётся с заднего разъёма с USART, далее можно через переходник в шину
- Проект создан в STM32CubeIDE с родным BSP и на FreeRTOS
- Сборка в STM32CubeIDE
- Морда отрисована и собрана в TouchGFX
- Написана Modbus либа и связана с интерфейсом и портом

Поиграться с интерфейсом можно в TouchGFX\build\bin



Проблемы и решения

1. PLC должен пинговаться из Fast, даже из виртуалки.
2. PLC должен быть в Run, проверьте, может он заклинился в Error
3. PLC должен открыть три порта 2000-2001-2002, можете терминалом проверить.
4. Обмен с PLC мониторится WaveShark и скриптом codac_spss_test.py
5. Fast должен поднять IOC, с DB с переменными @asyn, которые вам нужны
6. Эти же переменные должны быть в соответствующих DB в PLC, проверьте
7. Проверьте байт-смещение в DB IOC, оно должно биться с оным в DB в PLC
8. Asyn драйвер делает кучу проверок, и Fast рвёт связь если ему не понравилось, проверки такие:
   - таймаут на высылку пакетов как указано в cmd
   - строка версии в State должна совпадать
   - таймстемп в State съехать не более кажется 5 мин
   - alive counter должен инкрементально расти
   - *может ещё что, смотрите логи.*

Словарик

I&C	- вся система управления токамаком ITER
CODAC	- программная часть I&C, можно сказать дистрибутив, который ставится на ваш ПК
Fast	- просто ПК
SLow	- просто ПЛК
EPICS	- «ОС» для проведения научных экспериментов, программа с shared memory DB
EPICS PV	- переменная EPICS хранит некое значение, лежит в DB, привязана к времени, видна по IP сети
IOC	- экземпляр программы EPICS с конкретной DB который обслуживает конкретные переменные
s7asyn	- драйвер для связи IOC и PLC по IP
SPSS	- софт для PLC, который связывает его с IOC EPICS в CODAC
PSH	- отдельная виртуалка или просто процесс с IOC, например для поддержки обмена с Slow
SDD-Editor	- по сути конфигуратор DB для EPICS
CS-Studio	- визуализация переменных EPICS
CubeMon	- железка для мониторинга состояния кубикла на ИТЕР
TIA	- монструозная IDE для программирования PLC Siemens Simatic
Индус	- нехороший человек, редиска

Документация ITER

Полезно ознакомиться с плодовитыми писателями от PLC ITER.

CODAC PLC

ASYN_based_S7_PLC_driver.pptx
CCS_v6.2b5_s7asyn_PLC_driver_test_2AWLCR_v1_0.pdf
CODAC_HealthMon_Variables_35XFCY_v2_0.pdf
CWS_case_study_specifications_35W299_v4_2.pdf
Detailed_Technical_Specification_for_PLC_YNXRGU_v1_5.pdf
EPICS_-_S7_PLCs_Driver_Manual_PQYSSB_v1_0.pdf
EPICS_S7PLC_File_Specification-ITER_D_34_347QG5_v1_2.pdf
Guide_for_Development_of_the_Hardware_Ac_S9AV8C_v1_3.pdf
IO_Guidelines_for_PLC-Related_developmen_SPR9CK_v1_2.pptx
PLC_Integration_9BCH29_v2_7.pptx
PLC_Integration_Exercise__9NYD4G_v2_4.pdf
PLC_Sample_Users_Guide_2N8C3M_v4_2.pdf
PLC_Software_Engineering_Handbook_3QPL4H_v2_0.pdf
POC_Exercise_SLP7BQ_v1_7.pdf
s7PLCAsyn_EPICS_Driver_User's_Manual_PJAHXJ_v1_10.pdf
S7_PLC_EPICS_Driver_Extensions_6KFJEJ_v2_0.pdf
Siemens_Reference_List_AWYQ5G_v3_7.pdf
SPSS_Technical_Specification_6SRGGZ_v1_3.pdf
Standard_PLC_Software_Structure(SPSS)_Us_G4UMX5_v2_6.pdf

CubeMon (Integration Kit)

Kit_CubicleMon-SCH_QTD994_v1_6.pdf
Kit_CubicleMon-StartUp_RRF3NH_v2_0.pdf
Kit_CubicleMon-StartUp_RRF3NH_v2_2.pdf
Kit_Deployment_Guide_-_O_EBRQF5_v1_35.pdf
Kit_FC_NetGuide_JM3ETE_v1_7.pdf
Kit_Test_plan_GL6SX5_v1_36.pdf
Read_Me_First_Kit-CM_GL7SFH_v1_10.pdf
Read_Me_First_Kit-FC_JM5UED_v1_9.pdf
Read_Me_First_Kit-Net_KDSQPD_v1_8.pdf
Read_Me_First_Kit-SC_FQEHQ3_v1_16.pdf

По CODAC, SDD и CS-Studio отдельно и очень много

◄ EOF ►