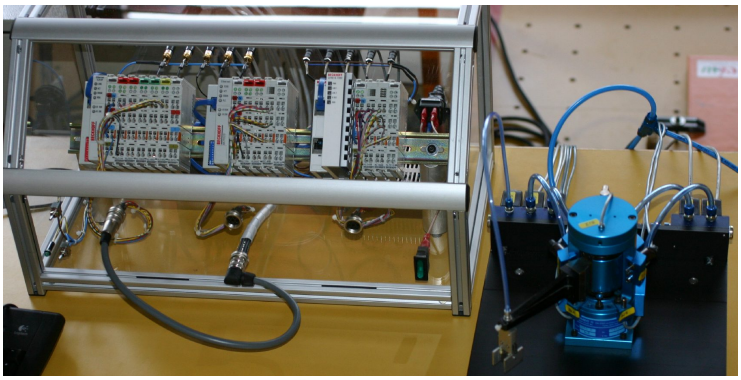
Cours de Robotique et d'Automatisation

3.2 Automate programmable industriel (API), Bus de terrain (et bras pneumatique : EXP. 25); Variantes 1131 – Modbus – Ethernet – TCP-IP (et Open : C++ ou Piaget ; sur PC)



HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD
www.heig-vd.ch

Institut d'Automatisation Industrielle iAi LaRA
Laboratoire de Robotique et Automatisation

Hes·so
Haute Ecole Spécialisée de Suisse occidentale

HESSO.HEIG-VD, iAi-LaRA, J.-D. Dessimoz, 13.04.2017

25 - Automate programmable industriel

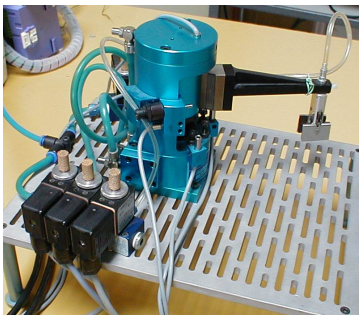
Contenu

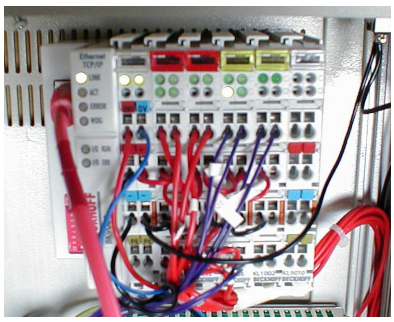
- Vue d'ensemble du matériel *Point express 1*
- Architectures possibles et recommandations *Point express 2*
- API (y c. Beckhoff)
- Norme IEC 61' 131 ("1131") pour la programmation d'API
- Norme Modbus – yc exemples
- Contexte C++
- Couche-applications et IHM (y c. Piaget et logiciel-métiers)
- Commande en C# (CS), avec réflexe géré en 1131 au niveau de l'API *Point express 3 … ou voir 3' (Annexe 4)*
- Commande en 1131 sur API BC 9020 *Point express 4 … ou voir 4' (Annexe 3)*
- Annexes

A1. Lancement et arrêt de la couche de communication TwinCAT sur PC
A2. Autres configurations proposées
A3. Application en 1131 - *Point express 4'*
A4. Application en C++ - *Point express 3'*
A5. Commande en Piaget - C# (CS) avec réflexe géré en 1131 au niveau de l'API

Point express 1

Vue d'ensemble du matériel





Bras à 2 mouvements (haut-bas et gauche-droite) et pince (ouverture ou non)

5 Capteurs inductifs+1 interrupteur;

5 électrovannes pour 2 vérins double effet et un piston pour la pince

Des entrées-sorties booléennes; donc une commande par API

L'API: de gauche à droite: accès Ethernet-TCP/IP; électronique; alimentation; 1 élément à 4 entrées, puis à 2 entrées; 1 élément à 4 sorties, puis 2 sorties; 1 élément terminal

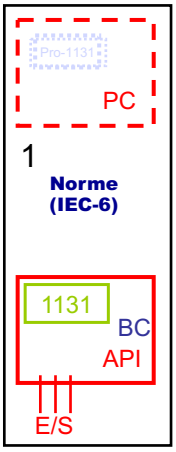
Point express 2

Voir aussi slide suivante?

Architectures possibles

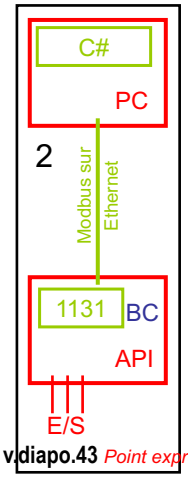
(et recommandations classiques)

Recommandation 2



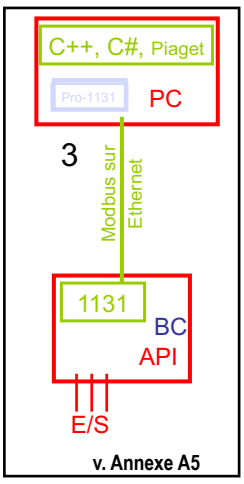
v.Diapo 48 *Point express 4*

Recommandation 1



v.diapo.43 *Point express 3*

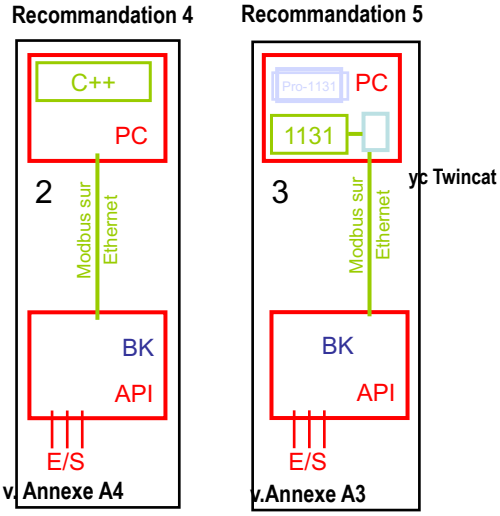
Recommandations, 3 (Piaget)



v. Annexe A5

No de diapo approximatif

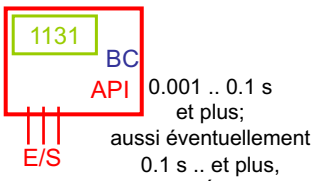
Architectures possibles
(et recommandations classiques)



API

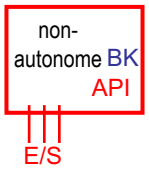
- Automate Programmable Industriel
- Exemple standard:

  Beckhoff BC 9000

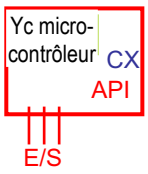


0.001 .. 0.1 s et plus;
aussi éventuellement 0.1 s .. et plus, sur réseau

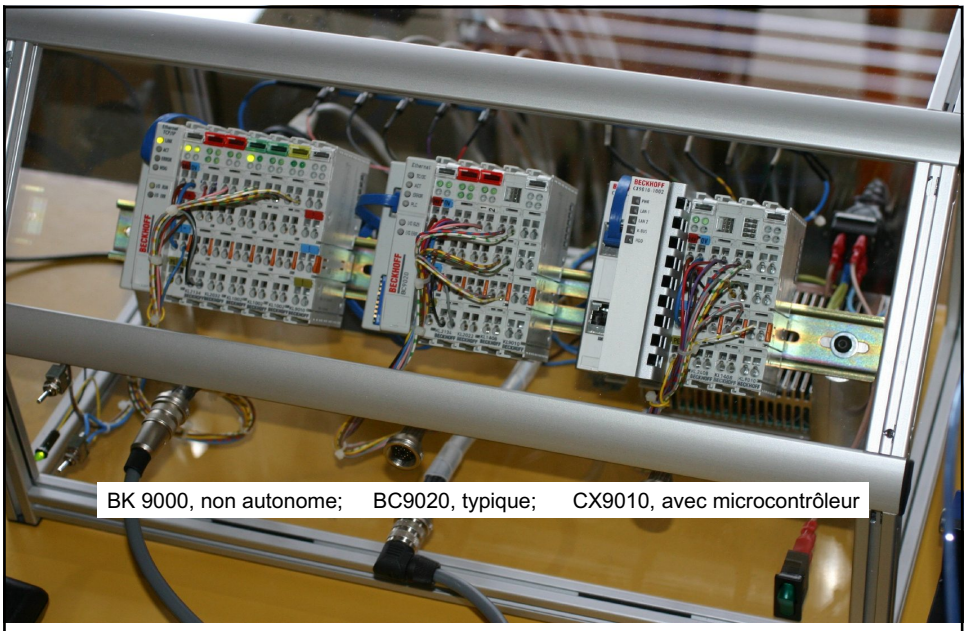
- Exemples spéciaux:



(0.001 .., si TwinCAT) 0.1 s .. et plus, sur réseau



0.000001 .. 0.001 s et plus;
et aussi comme ci-dessus



BK 9000, non autonome; BC9020, typique; CX9010, avec microcontrôleur

API - Module Beckhoff

Lors de ce laboratoire nous avons pu découvrir un système de bus de communication se basant sur Ethernet. Le module utilisé provient de la société Beckhoff. Ce système est actuellement utilisé à l'EIVD pour le projet de robot Hornuss. L'ensemble du système nerveux étant relié par Ethernet à l'aide de câbles blindés à paires torsadées. La distribution se fait en étoile, chaque partie étant reliée à un hub.

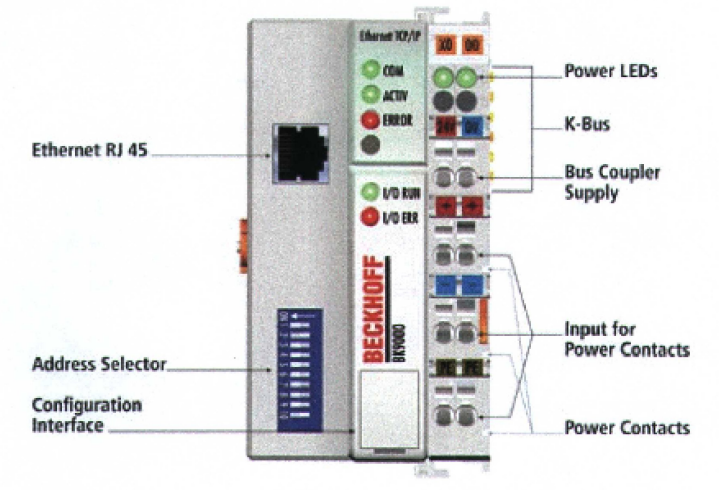
Les modules Beckhoff

Pour la gestion des entrées/sorties, le système Beckhoff BK9000 est utilisé. Il comporte un module de base servant de commande décentralisée. C'est cette partie qui va décodé les trames de protocole supérieure reçue et effectuer les actions adéquates. A ce module on couple des extensions permettant notamment de réaliser la commande TOR, PWM, l'acquisition analogique ou numérique etc. Une multitude d'extensions existent. Pour s'en rendre compte il suffit de passer sur le site de Beckhoff ! www.beckhoff.com

Pour ce laboratoire, un prototype de robot était à notre disposition. Sur celui-ci quelques capteurs (binaires) étaient reliés au BK9000. La connexion au PC de commande a été faite à l'aide d'un patch cord. Nous nous sommes directement branchés sur le hub présent sur le robot. L'utilité du hub se justifie notamment par la possibilité d'utiliser une caméra TCP/IP pour l'acquisition d'images décrivant l'environnement du robot.

tbritz/vnicod 11/05/2003

API - BK 9000



BK 9000 Technical data

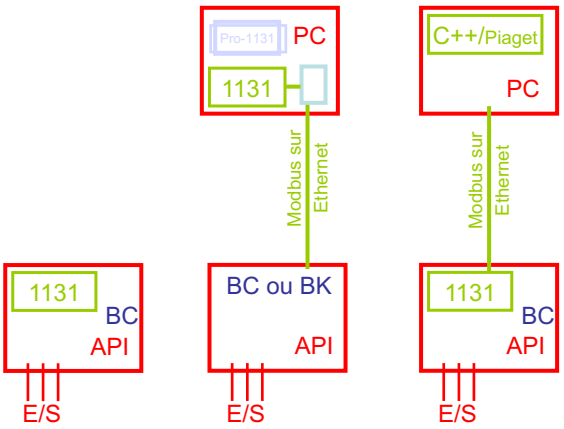
Technical data	BK9000
Number of Bus Terminals	64
Digital peripheral signals	256 inputs/outputs
Analog peripheral signals	128 inputs/outputs
Protocol	TwinCAT ADS, Modbus TCP
Baud rates	10/100 MBaud, automatic recognition of the transmission rate
Configuration possibility	KS2000 Software
Maximum number of bytes	512 bytes input and 512 bytes output
Bus connection	RJ 45
Power supply	24 V DC (-15%/+20%)
Input current	70 mA + (total K-Bus current)/4, 500 mA max.
K-Bus power supply up to	1750 mA
Starting current	2.5 x continuous current
Recommended fuse	≤ 10 A
Power contact voltage	24 V DC max.
Power contact current load	10 A max.
Dielectric strength	500 Vrms (power contact/supply voltage/fieldbus)
Weight approx.	170 g
Operating temperature	0 °C ... +55 °C
Storage temperature	-25 °C ... +85 °C
Relative humidity	95%, no condensation
Vibration/shock resistance	conforms to IEC 68-2-6/IEC 68-2-27
EMC resistance burst/ESD	conforms to EN 50082 (ESD, burst)/EN 50081
Protect. class/installation pos.	IP 20/variable

System data - Ethernet

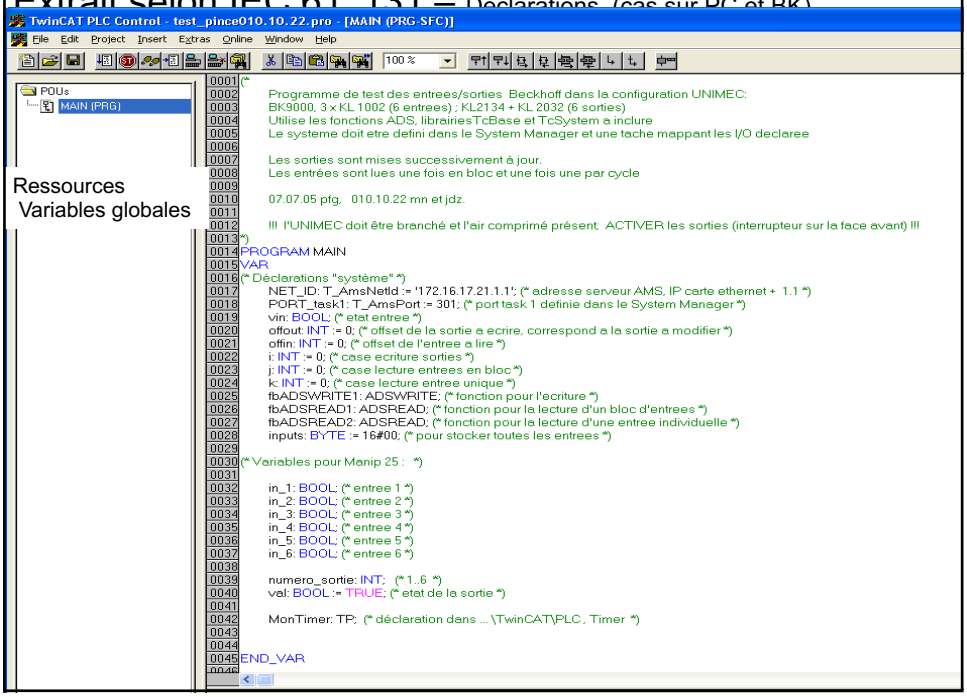
System data	Ethernet TCP/IP | BK9000
Number of I/O modules	only limited by the IP address space
Number of I/O points	depending on controller
Data transfer medium	4 x 2 twisted pair copper cable; category 3 (10 MBaud), category 5 (100 MBaud)
Cable length	100 m (distributor, hub to BK9000)
Data transfer rate	10/100 MBaud
Topology	star wiring

Norme IEC 61'131 pour la programmation d'API– yc Exemples

- Déclarations
- Code
  - IL
  - LD
  - FB
  - ST
  - Grafcet



Extrait selon IEC 61’131 – Déclarations (cas sur PC et BK)

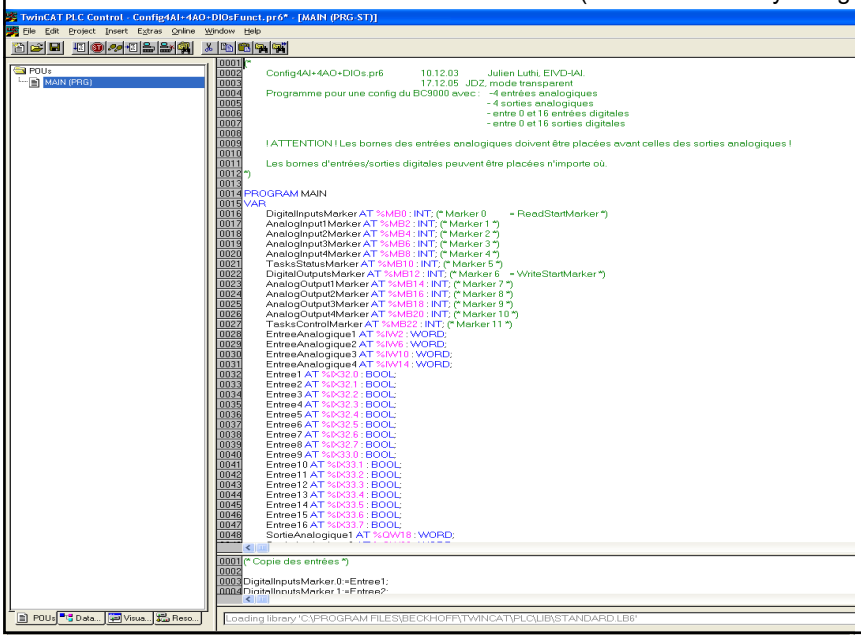


Extrait selon IEC 61’131
Déclaration concernant les E/S (cas sur BC autonome)

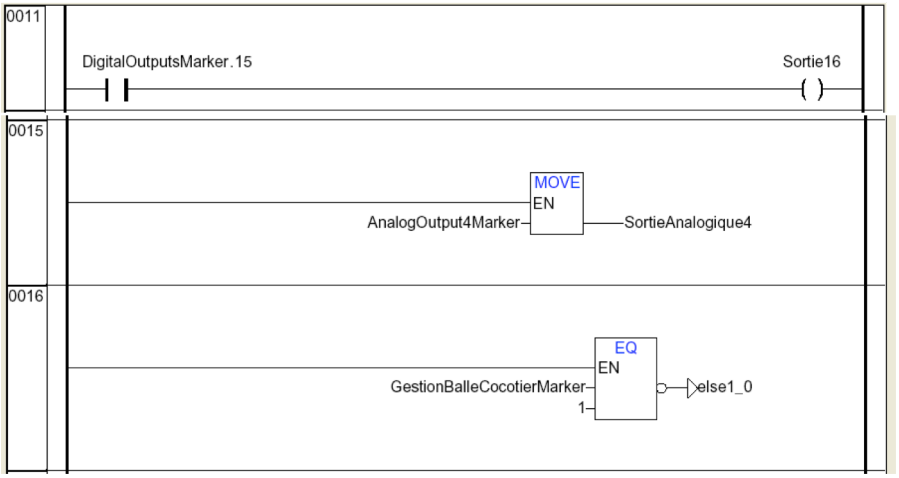
```
VAR
(* entrées *)
     CapteurGaucheBas     AT%IX32.0 : BOOL;
     CapteurGauche        AT%IX32.1 : BOOL;
     CapteurHaut          AT%IX32.2 : BOOL;
     CapteurDroit         AT%IX32.3 : BOOL;
     CapteurDroitBas      AT%IX32.4 : BOOL;

(* sorties *)
     Descendre       AT%QX32.0 : BOOL;
     Gauche          AT%QX32.1 : BOOL;
     Monter          AT%QX32.2 : BO
     Pince           AT%QX32.3 : BO
     Droite          AT%QX32.4 : BO
END_VAR
```

```
MAIN (PRG-SFC)
0001 PROGRAM MAIN
0002 VAR
0003     Entree2 AT %IX32.1:BOOL;
0004     Sortie3 AT %QX32.2:BOOL;
0005 END_VAR
0006
```

Extrait selon IEC 61’131
Déclaration concernant les E/S (cas sur BC + PC yc Piaget)



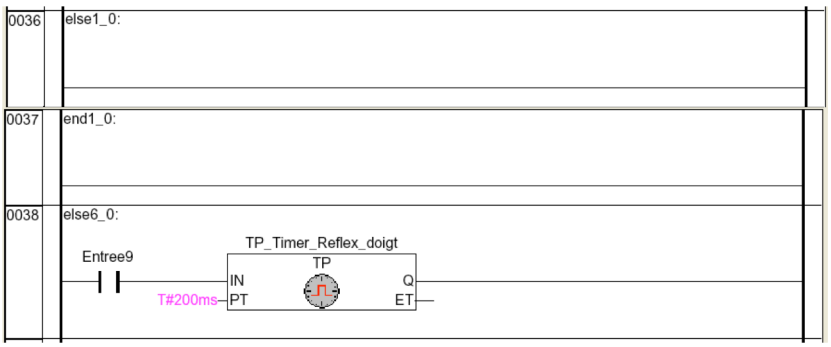
Extrait de code IEC 61’131
LD – Ladder Diagram



Extrait de code IEC 61’131

Ladder - suite

…



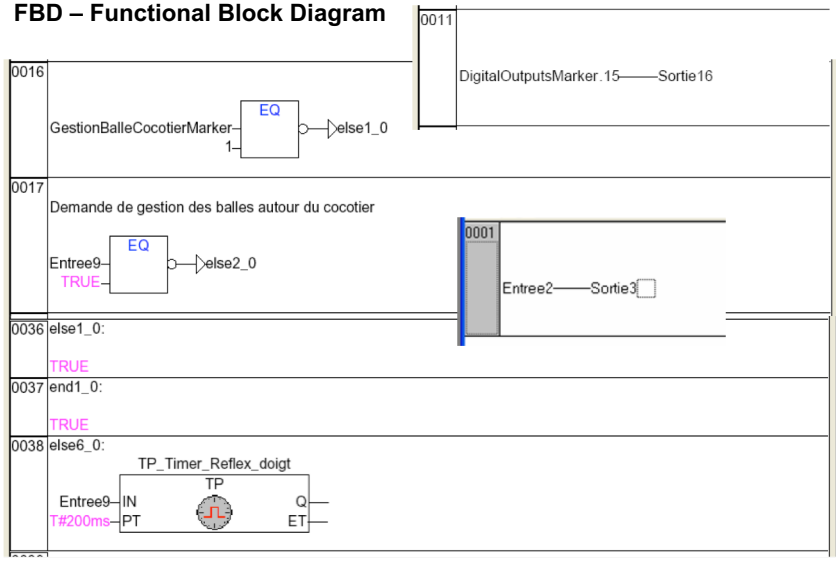
Extrait de code IEC 61’131

IL – Instruction List

```
0036 LD    DigitalOutputsMarker.15
0037 ST    Sortie16
0038
0039 LD    AnalogOutput1Marker
0040 ST    SortieAnalogique1
0041

0051 LD    GestionBalleCocotierMarker
0052 EQ    1
0053 NOT
0054
0055 JMPC      else1_0
0056

0124 else1_0:
0125 end1_0:

0126 CAL       TP_Timer_Reflex_doigt(IN := Entree9, PT := T#200ms)
0127
```

Extrait de code IEC 61’131

FBD – Functional Block Diagram



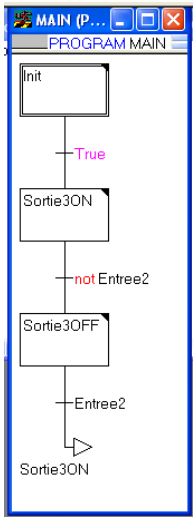
Extrait de code IEC 61’131 (3 de 6)

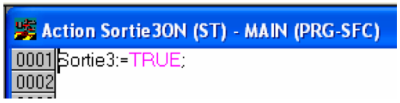
ST – Structured Text

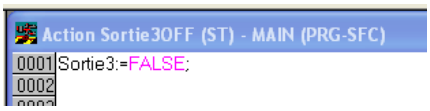
```
0022 SortieAnalogique4:=AnalogOutput4Marker;
0023
0024
0025 IF GestionBalleCocotierMarker = 1 THEN                    (*Demande de gestion des balles aut
0026     IF Entree9 = TRUE THEN
0027         IF (NBallePrisesMarker = 3) THEN                  (*Pas plus de balle*)
0028             Attendre_Fin_Gestion_Balle;
0029         ELSE
0030             IF NBallePrisesMarker = 0 THEN                (*Nb de balle dans le robot = 0*)
0031                 CMD_Prendre_Balle1_Du_Ciel := TRUE;
0032                 Var_Lancer_Gestion_Balle := TRUE;
0033             END_IF
0034             IF NBallePrisesMarker = 1 THEN                (*Nb de balle dans le robot = 1*)
0035                 CMD_Prendre_Balle2_Du_Ciel := TRUE;
0036                 Var_Lancer_Gestion_Balle := TRUE;
0037             END_IF
0038             IF NBallePrisesMarker = 2 THEN                (*Nb de balle dans le robot = 2*)
0039                 CMD_Prendre_Balle3_Du_Ciel := TRUE;
0040                 Var_Lancer_Gestion_Balle := TRUE;
0041             END_IF
0042         END_IF
0043     END_IF
0044 END_IF
0045
0046 TP_Timer_Reflex_doigt(IN:=Entree9,PT:=T#200ms,Q=>Sortie10);
0047
```

Extrait de code IEC 61’131

Grafcet







Extrait de code IEC 61’131

Grafcet

La réalisation des systèmes automatisés requiert, notamment, une description liant les effets aux causes. Pour cela, on décrira l'aspect logique du comportement souhaité du système.

La partie séquentielle du système désigne l'aspect logique d'un système physique auquel on accède par des variables d'entrée et des variables de sortie booléennes. Le comportement indique la manière dont les variables de sortie dépendent des variables d'entrée (voir note figure 1). Le GRAFCET a pour objet de spécifier le comportement de la partie séquentielle des systèmes.

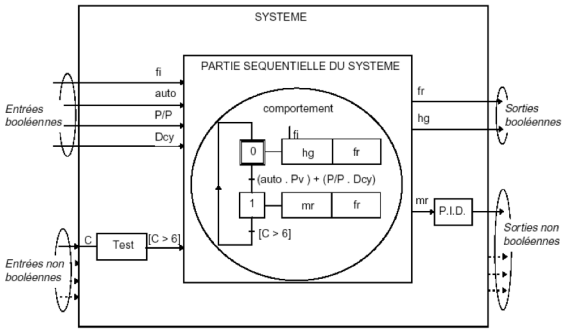


Figure 1- Représentation graphique de la partie séquentielle d'un système

NOTE : La partie séquentielle du système est caractérisée par ses variables d'entrée, ses variables de sortie, et son comportement. Cette partie séquentielle ne comporte que des variables d'entrées et de sorties booléennes, toutefois le langage de spécification GRAFCET permet par extension (exemple : évaluation d'un prédicat ou affectation d'une valeur numérique à une variable) de décrire le comportement de variables non booléennes.

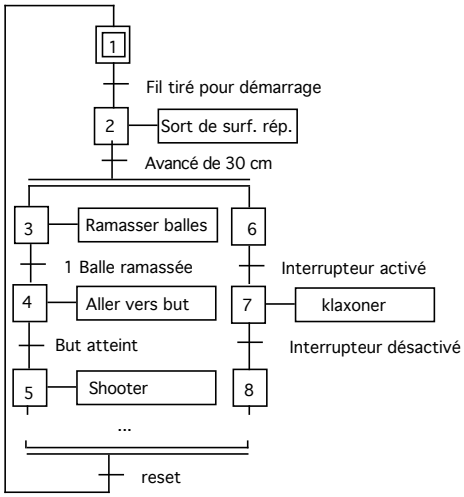
Extraits de la norme CEI 3B/256/CD de 1999

Extrait de code IEC 61’131

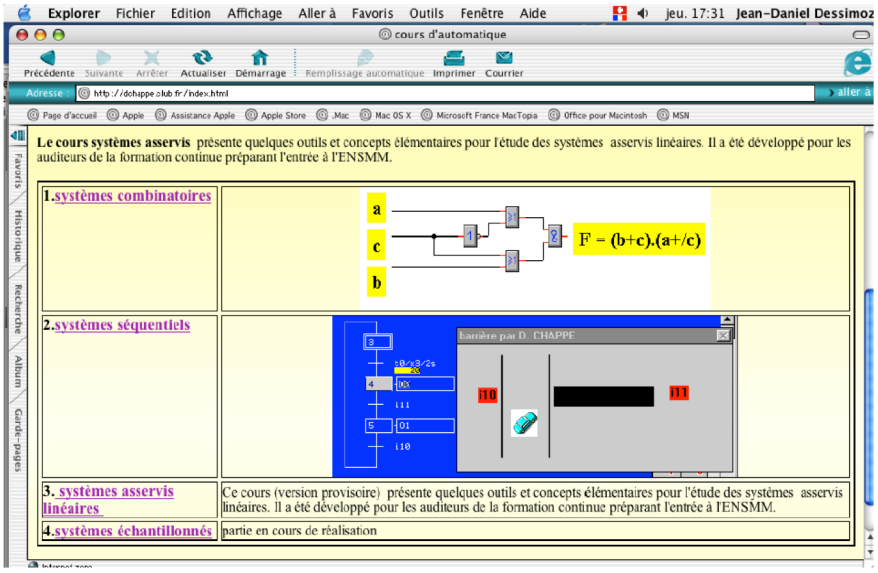
Grafcet

Spécification d’une tâche de commande en GRAFCET

Cas du robot footballeur



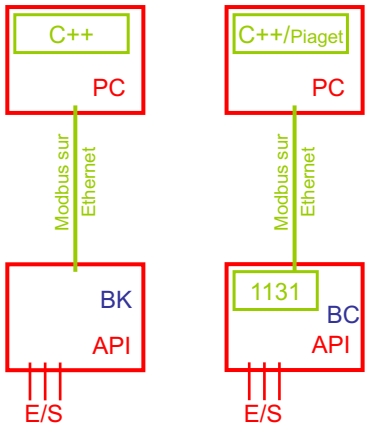
Exemple Grafcet en ligne



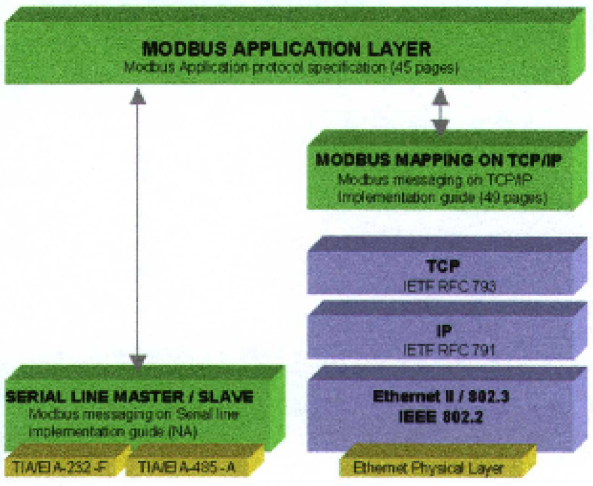
Norme Modbus - Exemples

Norme pour communication robuste, sur réseau de terrain (fieldbus)

- Couches physiques
- Télégrammes (en contexte C++)



Norme Modbus - Couches



tbritz/vnicod (extrait de www.modbus.org) : 11/05/2003

Communication Modbus

La faculté d'informer, en temps réel, les différentes parties commandes d'une chaîne automatisée, ne peut se faire qu'avec un réseau industriel de communication. Dans un tel réseau, les parties commandes sont reliées entre elles par des modules spécialisés utilisant des protocoles d'échanges bien définis, comme justement MODBUS.

Pour notre système de bus, TCP/IP ne sert que de couche de transport de l'information. En effet, il y a nécessité d'utiliser un langage (protocole de niveau supérieur) pour communiquer au module ce qu'il doit faire. C'est là qu'intervient Modbus.

Protocole

Le protocole MODBUS de GOULD MODICON est un protocole de type maître-esclave (un seul maître par réseau

Les échanges se font à l'initiative du maître, qui émet sa demande. Si l'esclave concerné l'a comprise, il renvoie sa réponse.

Les fonctions du maître peuvent se résumer en ces quelques points:

- Assurer l'échange des données entre "esclaves". Les "esclaves" ne pouvant dialoguer entre eux, le maître assure le passage des différentes informations.
- Assurer un dialogue évolué avec un opérateur (dialogue homme/machine). Le "maître" est en général pourvu d'un écran graphique (PC) qui permet de visualiser de façon dynamique l'évolution du processus en cours.

tbritz/vnicod (extrait de www.modbus.org) : 11/05/2003

Quelques fonctions Modbus

Code (cf octet No 7 des télégrammes Modbus) et description:

Entrées-sorties directes(cas typique avec BK9000):

2 : lire entrée digitale
4 : lire entrée analogique
15 : écrire sortie digitale

Dialogue avec ordi local (cas typique avec BC 9000) :

23 : Lire ou écrire des variables (synonymes ici: markers ou registres)

Extrait de code C++ décrivant les télégrammes de gestion des E/S selon la norme Modbus - Header

```
//---------------- Extraits de Hornuss
// environnemment Piaget, implémenté en C++, pour la commande embarquée, multiagents et temps-réel du robot mobile
autonome Hornuss  (copyright HESSO-HEIG, 2003-2005)
…
//---------------- Extraits de uTEthernet.cpp  (copyright HESSO-HEIG, 2003-2005
const AnsiString BeckAddress = "172.16.17.1" ;
//const AnsiString BeckAddress = "172.16.17.6" ;
…
//------------- Extraits de uThreads.cpp  (copyright HESSO-HEIG, 2003-2005
void __fastcall ThBeck::Execute( )
{
  byte Message [25];
  int i,j,Entree,val,tmp,val2 ;
  char buffer[10];
  char buf[10];
  char lecture[25];
  int Fonction;
  TWinSocketStream *pStream ;
      switch (FThBeckHoff) { case
//////////////////////////////
-
```


Extrait de code C++ décrivant les télégrammes de gestion des E/S selon la norme Modbus - EntréesD

```
//////////////////////////////
   EntreesD:  ;
     pStream = new TWinSocketStream( Form1->CSBeck1->Socket, 1000);
     Message [1]  = 0;
     Message [2]  = 0;   // toujours 0
     Message [3]  = 0;   // toujours 0
     Message [4]  = 0;   // 0 si < que 256 bytes
     Message [5]  = 6;   // Nb de byte qui suivent
     Message [6]  = 1;   // UnitÉ
     Message [7]  = 2;   // Fonction Modbus: lire entrÉe digitale
     Message [8]  = 0;   // Adresse de dÉpart "high"
     Message [9]  = 0;   // Adresse de dÉpart "low"
     Message [10]  = 0;
     Message [11]  = NSignauxBoolIn ;  // Nb d'entrÉe à lire paramètre du programme
         for(i=0; i<12; i++)
     {
         buffer[0] = Message[i];
       pStream->Write(buffer,1);
     }
-
```

Extrait de code C++ décrivant les télégrammes de gestion des E/S selon la norme Modbus - Lecture

```
// Lecture
memset(lecture, 0, 25);      // initialisation du buffer
pStream->Read(lecture, 25);  // lecture
Fonction = lecture[7]; // fonction modbus
   if (Fonction == 2)      // Si la fonction Modbus est la lecture
   {
        Entree = lecture[9];    // NumÉro des entrÉes
  for (i=1;i<=NSignauxBoolIn;i++)
  {
   if ((Entree & (int)pow(2,i-1 )) == (int)pow(2,i-1 ))
          SignauxIn[i].EtatVF  = true ;
     else
          SignauxIn[i].EtatVF  = false;
     if( SignauxIn[i].AComplementer)
       SignauxIn[i].EtatVF=!SignauxIn[i].EtatVF;
//        SignauxIn[i].actif= true ;
  }
}
delete pStream;
Work[6].TaskStatus = Faite;
break; case
//////////////////////////////
```


Extrait de code C++ décrivant les télégrammes de gestion des E/S selon la norme Modbus - SortiesD

```
SortiesD: ;
   pStream = new TWinSocketStream( Form1->CSBeck1->Socket, 1000);
   //////////
   // ATTENTION si NSignauxBoolOut > 8 ajouter d'autres bytes
   //////////
   val = 0 ;
   tmp = 1 ;
   //  for(i=8;i<=NSignauxBoolOut;i++)
   for(i=1;i<=8;i++) {
        if (SignauxOut[i].EtatVF)
      val = val + tmp ;
    tmp = tmp * 2 ;
   }
   val2 = 0 ;
   tmp = 1 ;
   for(i=9;i<=NSignauxBoolOut;i++) {
        if (SignauxOut[i].EtatVF)
      val2 = val2 + tmp ;
    tmp = tmp * 2 ;
   }
   Message [1]  = 0;
   Message [2]  = 0;    // toujours 0
   Message [3]  = 0;    // toujours 0
```

Extrait de code C++ décrivant les télégrammes de gestion des E/S … - SortiesD suite

```
Message [4]  = 0;    // 0 si < que 256 bytes
Message [5]  = 9;    // Nb de byte qui suivent
Message [6]  = 1;    // UnitÉ
Message [7]  = 15;   // Fonction Modbus: Écrire sortie digitale
Message [8]  = 0;    // Adresse de dÉpart "high"
Message [9]  = 0;    // Adresse de dÉpart "low"
Message [10] = 0;    // Length high
Message [11] = 10;    // Length low (nb de sorties)
Message [12] = 2;    // Nb de byte pour envoyer les xx sorties
Message [13] = val;  // byte de sorties
Message [14] = val2;  // byte de sorties
/*   Message [12] = 1;    // Nb de byte pour envoyer les xx sorties
  Message [13] = val;  // byte de sorties   */
  for(j=0; j<15; j++)
  {  buffer[0]= Message[j] ;
     pStream->Write(buffer,1);
  }
  memset(lecture, 0, 25);      // initialisation du buffer
  pStream->Read(lecture, 25);
  delete pStream;
  Work[6].TaskStatus = Faite;
                                        break; case
//////////////////////////
```


Extrait de code C++ décrivant les télégrammes de gestion des E/S selon la norme Modbus - EntréesA

```
EntreesA: ;
  pStream = new TWinSocketStream(Form1->CSBeck1->Socket, 1000);
  Message [1]  = 0;
  Message [2]  = 0;   // toujours 0
  Message [3]  = 0;   // toujours 0
  Message [4]  = 0;   // 0 si < que 256 bytes
  Message [5]  = 6;   // Nb de byte qui suivent
  Message [6]  = 1;   // UnitÉ
  Message [7]  = 4;   // Fonction Modbus: lire entrÉe analogique
  Message [8]  = 0;   // Adresse de dÉpart "high"
  Message [9]  = 0;   // Adresse de dÉpart "low"
  Message [10]  = 0;
  Message [11]  = NSignauxAnalogiquesIn*2;  // Nb d'entrÉe à lire
 for(i=0; i<12; i++)
  {
     buffer[0] = Message[i];
     pStream->Write(buffer,1);
  }
  //////Lire
  memset(lecture, 0, 60);
  pStream->Read(lecture, 60);   //nb de bytes reçus
```

Extrait de code C++ décrivant les télégrammes de gestion des E/S ... - EntréesA suite

```
//////Lire
        memset(lecture, 0, 60);
        pStream->Read(lecture, 60);   //nb de bytes reçus
        if (lecture[7] == 4)
        {
          SignauxIn[NSignauxBoolIn+1].valeur =
            (float)(lecture[11]*256 + lecture[12]) * (float)1 / (float)32767 ;
          SignauxIn[NSignauxBoolIn+2].valeur =
            (float)(lecture[15]*256 + lecture[16]) * (float)1 / (float)32767 ;
          SignauxIn[NSignauxBoolIn+3].valeur =
            (float)(lecture[19]*256 + lecture[18]) * (float)1 / (float)32767 ;
          SignauxIn[NSignauxBoolIn+4].valeur =
            (float)(lecture[23]*256 + lecture[22]) * (float)1 / (float)32767 ;
        }
delete pStream;
        Work[6].TaskStatus = Faite;
        break;
/////////////// Autre tâches //////////////
    default: ;
  }
}
```


Extrait de code C++ décrivant les télégrammes de gestion des E/S ... - Echange de variables (1de 2)

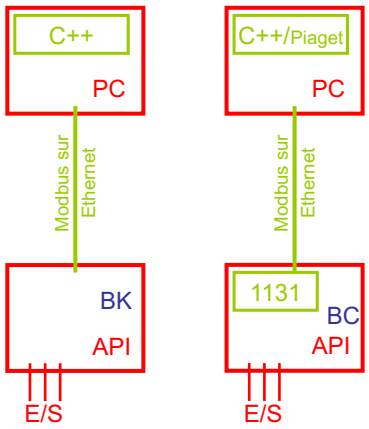
```
void BC9000::ReadWriteRegisters( byte*   readAddress,
    byte readN, byte* writeAddress, byte writeN,byte* data)
{//fonction modbus 23, ne pas modifier
 try
 {modbusMessage[0]  = 0; //transaction identifier, returned
                           by slave
  modbusMessage[1]  = 0; //transaction identifier, returned
                           by slave
  modbusMessage[2]  = 0; //protocol identifier, always 0
  modbusMessage[3]  = 0; //protocol identifier, always 0
  modbusMessage[4]  = 0; //length field
  modbusMessage[5]  = 11+2*writeN;//number of following bytes
  modbusMessage[6]  = 1; //unit
  modbusMessage[7]  = 23; //modbus function : read/write
                           registers
  modbusMessage[8]=readAddress[0];//read start address high
  modbusMessage[9]=readAddress[1]; //read start address low
```

Extrait de code C++ décrivant les télégrammes de gestion des E/S ... - Echange de variables (2 de 2)

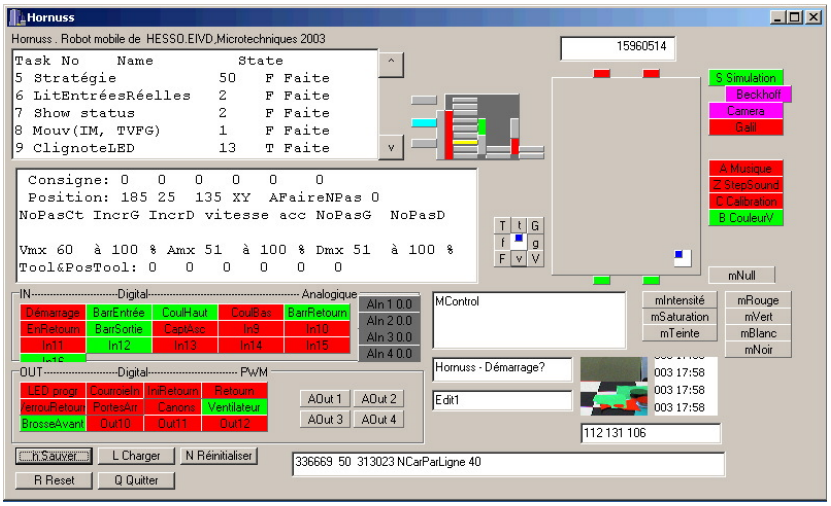
```
modbusMessage[10] = 0x00;    //read length high
modbusMessage[11] = readN;   //read length low
modbusMessage[12]=writeAddress[0];//write start address high
modbusMessage[13]=writeAddress[1]; //write start address low
modbusMessage[14] = 0x00;    //write length high (words!)
modbusMessage[15] = writeN;  //write length low (words!)
modbusMessage[16] = 2*writeN;//byte count
for( int i = 0; i < (2*writeN); i++ )
 {if( ((17+i) >= MAX_MESSAGE_SIZE) || (i > 13) )
    {ShowMessage( "Debordement 1 ¥n");
    return; }
  else
   {modbusMessage[17+i] = data[i];}  } // end for
modbusMessageSize = 17+(2*writeN);
SendMessage();
GetMessage();
}
```


Couche-applications et IHM

- Interface Homme-Machine (ex. Hornuss)
- Piaget
- Contexte C++ et autres logiciels-métiers



Interface Homme-Machine pour Hornuss, dont la majorité des E/S se font par API et Modbus sur Ethernet



Piaget - « Tâche 11 – Réflexe du doigt »

```
//////////////////////////////////////////////////////////////////////
//////////////////////   11 11 11 11 11 11 11 11 11 ////////////////////////
//////////////////////////////////////////////////////////////////////
 void Task11() // RÈflexe doigt
{ int const TaskNumber=11;
  int none;
  ActiveTask=TaskNumber;
 switch ( Work[TaskNumber].StateNo ){
 case
     1 :
         if (Work[TaskNumber].TaskStatus == Demandee)
            Work[TaskNumber].StateNo = 3;
                                        break;case
   3: GoNext();
                                        break; case
   4:   if (SignalActifEtVrai(NSICapteurBalleCoco1))
              {       Work[TaskNumber].TaskStatus = EnAction;
                      Work[NTStrategie].TaskStatus = Suspendue;
                      Work[TaskNumber].StateNo = 5;  }
            else
                      Work[TaskNumber].StateNo = 1;
                                        break;case
```

Piaget - « Tâche 11 – Réflexe du doigt »

```
5:    if (NBallesPrises==3)   // pas plus de balles
            GoState(20);
      else GoNext();
      break;case
6:    switch (NBallesPrises) { case
            0:LancerGestionBallesAGN(CMDPrendreBalle1DuCiel);
                  break; case
            1:LancerGestionBallesAGN(CMDPrendreBalle2DuCiel);
                  break; case
            2:LancerGestionBallesAGN(CMDPrendreBalle3DuCiel);
                  break;
            default: GoState(20);
      }
      break; case
7:    GoNext();
      break; case
8:    SignalOutAGN(NSOSortirDoigt, true);
      break;case
```


Piaget - « Tâche 11 – Réflexe du doigt »

```
9:    SleepAGN(0.2);
                              break;case
10:
      SignalOutAGN(NSOSortirDoigt, false);
                              break;case
11:   SleepAGN(1.5);
                              break; case
12:   Work[TaskNumber].StateNo = 20;
                              break;case
20:   Work[NTStrategie].TaskStatus = EnAction;
      GoState(30);
                              break;case
30:
      Work[TaskNumber].TaskStatus = Faite;
      Work[TaskNumber].StateNo = 1;
                               break;
 default :
   {
    MessageErreur="Task"+IntToStr(TaskNumber)+" - Line missing:
"+IntToStr(Work[TaskNumber].StateNo);
  };
 }
}
```

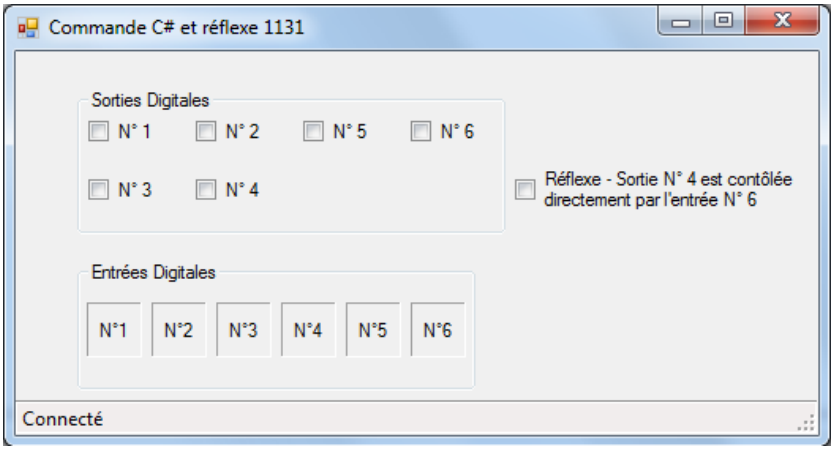
Point express 3

Commande en C# sur PC, avec réflexe en 1131, sur l'API BC 9020 (1 de 5)

- Le Manutec doit se brancher sur l'API Beckhoff BC9020
- Copier le répertoire, faire tourner, puis modifier le code écrit en C# :
  « D:\25-Code at D\BC9020\BC9020 C# - Modbus - yc réflexe\std - C# - Modbus - BC9020 - 016.11.03 »; (cliquer sur le fichier … .sln, puis sur le bouton "start" – trangle vert).
- Programmer un bouton pour fermer la pince, puis une suite d'actions
- 3 « tuyaux »:
  - L'interrupteur « Sorties Actives » doit être enclenché
  - Test d'une entrée: if(label6.BackColor==ColorLimeGreen)…
  - Activation d'une sortie: CheckBox5.Checked=true;
  - Attente: cf. System.Threading.Thread. Sleep(500); // ms

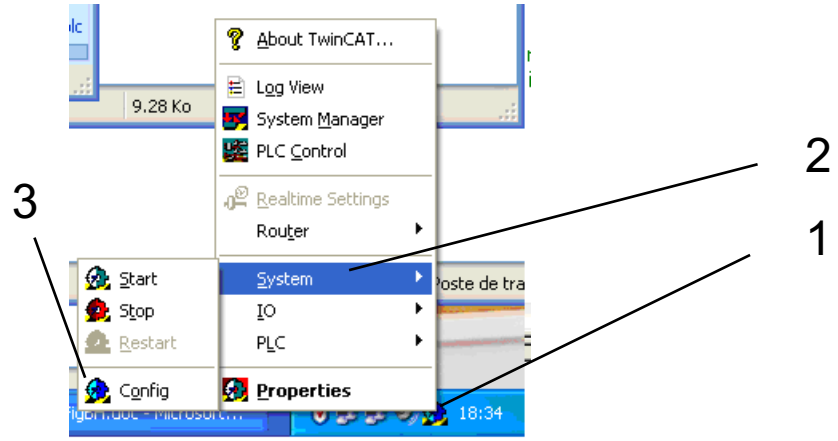
* En cas de problème:
- L'état de Twincat est de préférence "Config" pour l'utilisation du BC 9020
- Charger dans l'API, si nécessaire, (yc Create boot project) le code 1131 du réflexe: « D:\25-Code at D\BC9020\BC9020 C# - Modbus - yc réflexe\BC9020 1131 for Cpp or C# and Grafcet 016.11.03- std\BC9020_config_std.prx»;
- Dans le même répertoire, cliquer le fichier .tsm (puis Choisir système cible, et Action/Activer configuration; v.a. slide 4 et 5 de 5)

Commande en C# sur PC, avec réflexe en 1131, sur l'API BC 9020 (2 de 5)



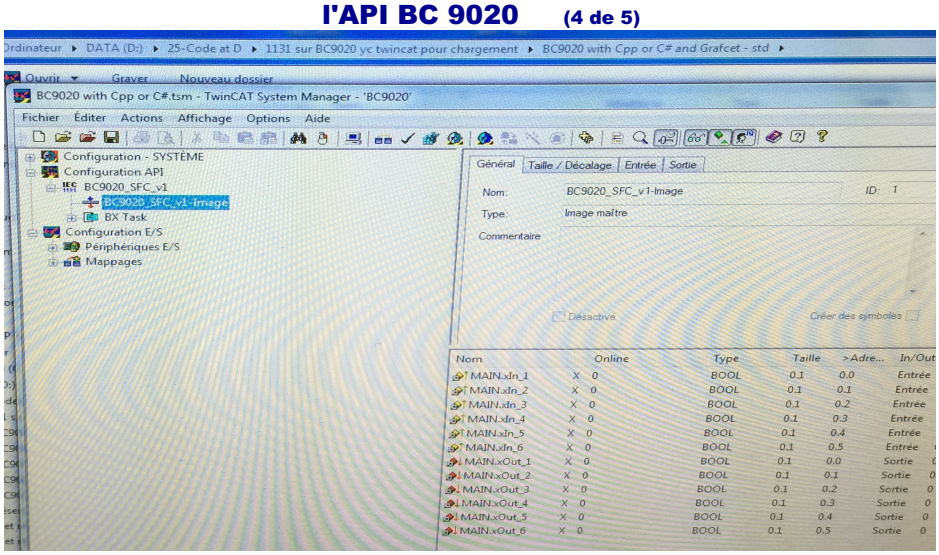
25 - Automate programmable industriel

Commande en C# sur PC, avec réflexe en 1131, sur l'API BC 9020 (3 de 5)



25 - Automate programmable industriel

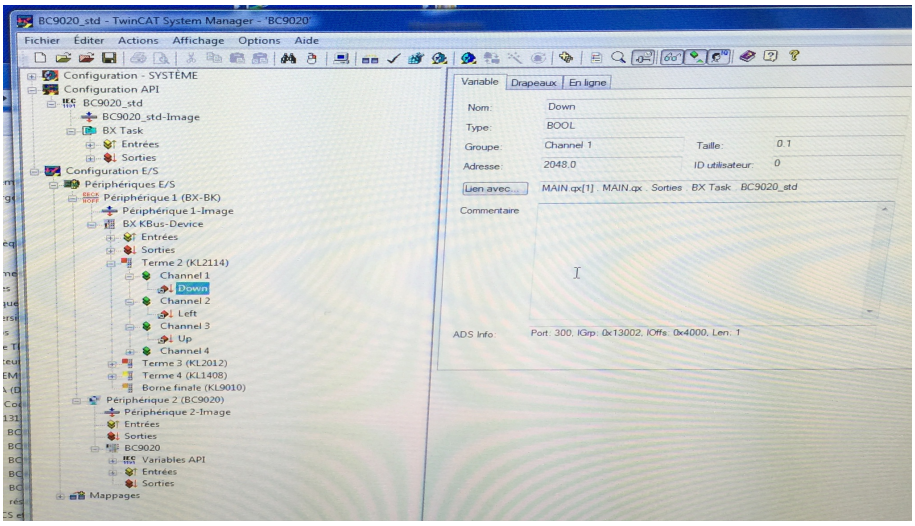
Commande en C# sur PC, avec réflexe en 1131, sur l'API BC 9020 (4 de 5)



- La configuration doir s'associer à un programme; en principe, c'est prêt.

25 - Automate programmable industriel

Commande en C# sur PC, avec réflexe en 1131, sur l'API BC 9020 (5 de 5)



- La configuration lie les E/S aux variables; en principe, c'est prêt.

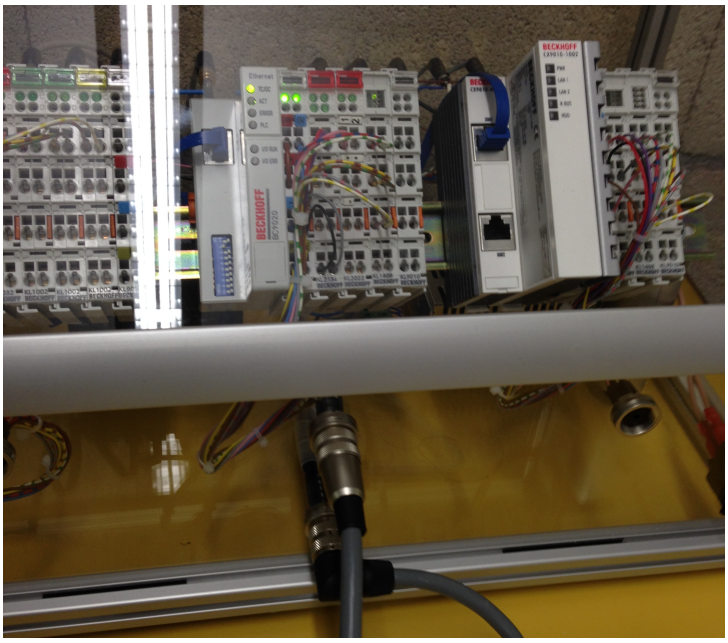
25 - Automate programmable industriel

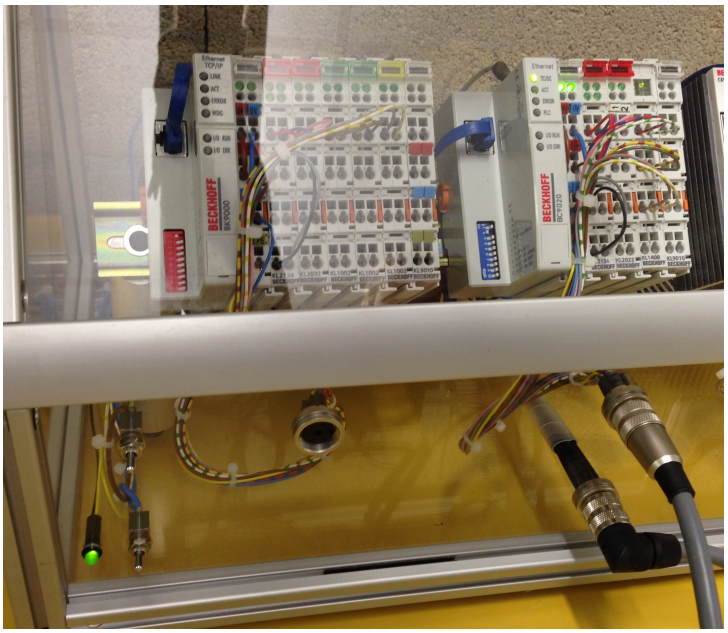
Point express 4

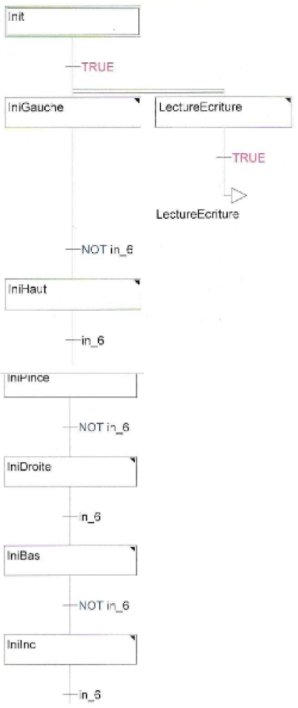
Commande en 1131 sur API BC 9020

- Le Manutec doit se brancher sur le BC9020
- Observer que la commutation de l'interrupteur change l'état de la pince
- Twincat doit être actif (cf Annexe 1)
- Copier le répertoire, faire tourner, puis modifier, sur D:,
  « D:\25-Code at D\BC9020\BC9020 C# - Modbus - yc réflexe\BC9020 1131 for Cpp or C# and Grafcet 016.11.03- std\BC9020_config_std.prx»; (à l'exécution: sélectionner Online\login, Online\run).
- Programmer une suite d'actions
- 4 « tuyaux »:
  - Si on veut changer de façon durable le code de l'API, cliquer Online\ Create Boot project
  - Il existe aussi une version avec Grafcet
  - L'interrupteur « Sorties Actives » doit être enclenché
  - Un temps minimum d'attente peut s'imposer à une étape: clic droit, choix "Time overview", ex.: t#3s pour 3 secondes.
  - ou:

```
TP_Timer_Reflex_doigt(IN:=Entree9,PT:=T#200ms,Q=>Sortie10);
```


BECKHOFF
BECKHOFF
BC9020


BECKHOFF
BECKHOFF
BC9020



Essais en programmation 1131

(2 de 3)

- Application en Grafcet (entrée 6=interrupteur) par F. Maiorana et B. Rochat, 2008

Essais en programmation 1131 (3 de 3)

- Exemple d'étape en Grafcet avec timer par Corentin Magnenat et Max Manier, 2008

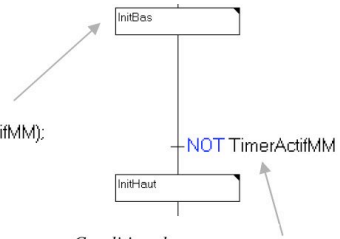
Déclaration:

```
TP_TimerMM:TP; (*Timer de Manier Magnenat*)
TimerActifMM: BOOL := FALSE;(*Variable de Manier Magnenat*)
```

Utilisation:

Appel du timer (ici : durée 1s), tant que le timer est actif, la variable TimerActifMM vaut TRUE.

```
TP_TimerMM(IN:=NOT TimerActifMM, PT:=T#1000ms, Q=>TimerActifMM);
```



Condition de passage au bloc suivant : TimerActifMM = FALSE

Ou simplement (JDZ 010.11.05):

```
MonTimer : TP; (* déclaration *)
MonTimer(PT:=T#1s); (* initialisation: durée à l'état « 1 » *)
MonTimer(IN:=false); MonTimer(IN:=true); (* reset, set *)
MonTimer.Q  (* variable de sortie, comme condition de transition *)
```

Conclusion

- Les architectures varient, du simple au plus complexes
- Les API Beckhoff ont été parmi les tout premiers à être intégrables avec la norme Ethernet et TCP/IP
- Les IEC 61'131 définissent plusieurs modes de programmation (IL, LD, FB, ST, Grafcet)
- Modbus est un bus de terrain courant pour API
- La tendance est à la commande par ordinateur. Le contexte C++ est particulièrement adapté pour de tels développements.
- En pratique, une couche-applications et un IHM adaptés sont souhaitables (ex.: Piaget et logiciel-métiers)
- Des environnements de programmation pour 1131 existent sur PC, même si la cible est ensuite autonome (ex.: TwinCAT)
- Attention à ne pas avoir de conflit d'accès aux ressources (ex. couche de communication par dll simultanément au dialogue Modbus, entre PC et API)

Annexes

A1. Lancement et arrêt de la couche de communication TwinCAT sur PC

A2. Autres configurations proposées

A2. Commande avec C#, BC 9020, et réflexe (écran)

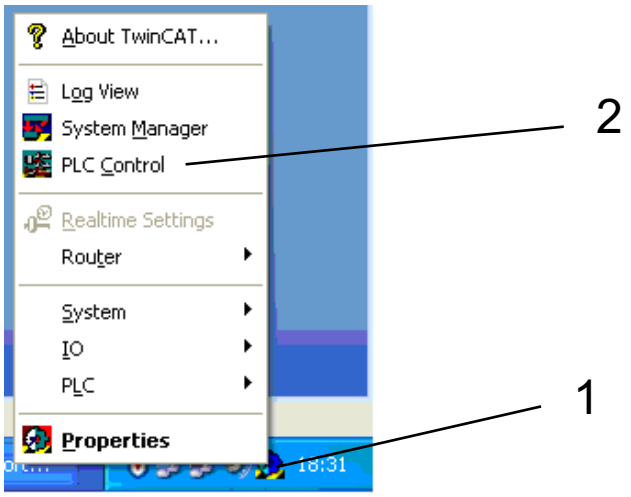
Annexe 1

A1. Lancement et arrêt de la couche TwinCAT sur PC 1 de 4

- Suivant l'architecture retenue, il peut-être indispensable, ou au contraire rédhibitoire, d'utiliser sur PC une couche de communication spéciale, rapide, entre API extérieur et processus internes (DLL, tournant en "background", pour TwinCAT).
  - Dans le cas du 1131 tournant sur PC, avec BK, la couche de communication doit s'enclencher (« start » ci-après), et ceci permet des temps de réaction de 1 ms.
  - Dans les autres cas, la couche doit se déclencher (« stop » ci-après). On utilise alors la communication Ethernet TCP/IP ordinaire, avec 0.1 ms de temps en entrée ou sortie du PC (c'est alors, typiquement, l'API qui assure les réactions de l'ordre de la milliseconde).

Annexe 1

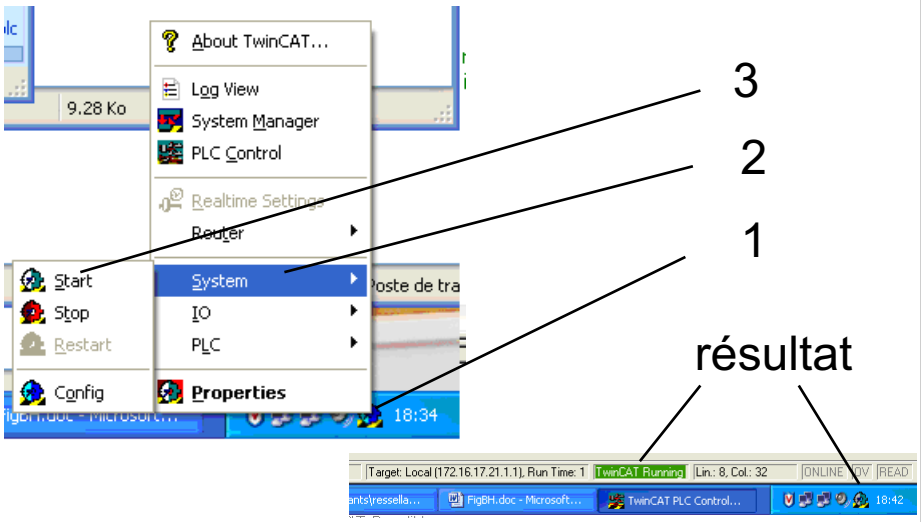
A1. Lancement et arrêt de la couche TwinCAT sur PC 2 de 4



Annexe 1

START

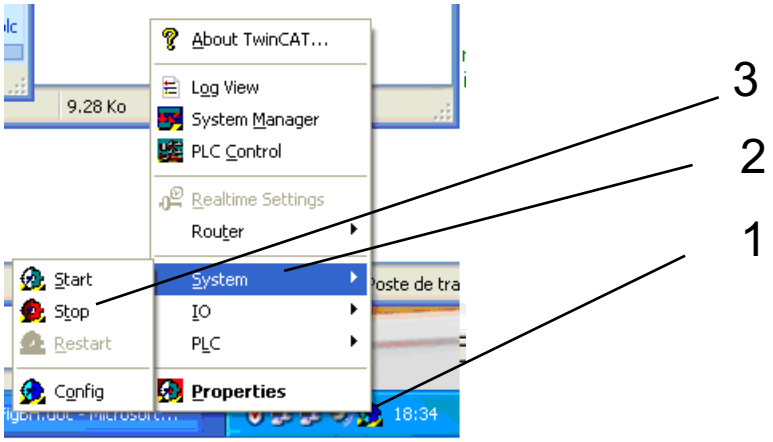
A1. Lancement et arrêt de la couche TwinCAT sur PC 3 de 4



Annexe 1

STOP

A1. Lancement et arrêt de la couche TwinCAT sur PC 4 de 4

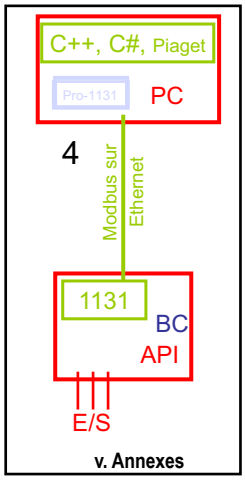


Annexe 2

A2. Autres configurations proposées 1 de 2

- **A3. Commande en 1131 sur API BC 9020 (configuration traditionnelle typique pour API)**
- **A4. Commande en C# (CS) avec réflexe géré en 1131 au niveau de l'API (configuration moderne, intégrant la supervision sur PC , et le fonctionnement classique sur l'API)**
- **A5. Commande en Piaget - C# (CS) avec réflexe géré en 1131 au niveau de l'API (configuration LaRA, intégrant de nombreuses ressources pour robots et cognitique)**

Recommandations $1b_i$, $2b_i$, 3
(éventuels C#, réflexe, Piaget)



Annexe 2

A2. Autres configurations proposées 2 de 2

Répertoires correspondants

Racine commune (dans "Manip…" ou sur "disque D:") :

• \\eistore2\iai-lara\ManipulationsLaRA\25 API-Modbus-Ethernet-P&P\Labo 25 - Code

Répertoires spécifiques:

•1131 sur BC9020 yc twincat pour chargement	**- A3.**
•Cpp et réflexe 1131 sur BC9020 - 015.03.05	**- similaire à A4**
•CS et réflexe 1131 sur BC9020	**- A4.**
•Graphcet - voir 1131 sur BK; ou 1131 dans A3	**cf dernière ligne ou A3**
•Piaget CS with reflex 1131	**- A5.**
•UNIMEC-TwinCAT 1131 et C++ sur BK9000	**- Cas traditionnels (points express 3 et 4)**

Annexe 3

Essais en programmation 1131

- Le Manutec doit se brancher sur le BK9000
- Twincat doit être actif (cf. Annexe 1)
- Faire tourner, puis modifier (une copie de) , sur D:,
  « 25-Code\UNIMEC-TwinCAT 1131 et C++ sur BK9000\Twincat\1131\Sur PC avec couche Twincat en Background Et BK\1131SurPCAvecDLLNewTestPince yc ppt\test-pince.pro »
- Programmer une suite d'action
- 9 tuyaux:
  - Utiliser l'interrupteur pour commander l'avance de la séquence
  - Pour effacer une étape il faut sélectionner ensemble une étape et une transition
  - Pour lancer l'exécution: Online/login puis Online/Run
  - Possibilité d'ajouter un timer (voir ci-après)
  - L'interrupteur « Sorties actives » doit être enclenché (pos. haute)
  - Voir aussi la présentationTwinCat…test_pince.pro.ppt
  - Déclarations visibles en slide 11
  - Dans une étape, une seule sortie peut être changée
  - Un temps minimum d'attente peut s'imposer à une étape: clic droit, choix "Time overview", ex.: t#3s pour 3 secondes.

Annexe 4

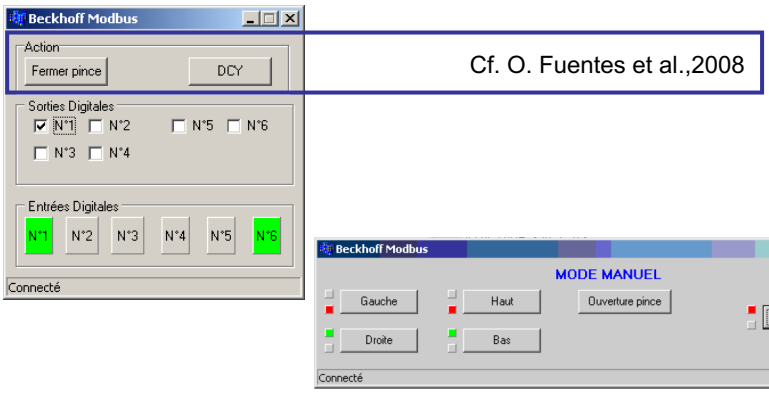
A4. Essais en programmation C++ (1 de 2)

- Le Manutec doit se brancher sur le BK9000
- Twincat doit être inactif (cf Annexe 1)
- Copier le répertoire Std, faire tourner, puis modifier, sur D:,
  « 25-Code\UNIMEC-TwinCAT 1131 et C++\Cpp sur BK9000\Std\Modbus.bpr »
- Ajouter un bouton « Fermer pince » (cf. fascicule – Pour débuter en C++)
- 5 « tuyaux »:
  - Possibilité d'utiliser l'instruction suivante:
    CheckBox4->Checked=true; // active sortie 5
  - Le code x=(condition)?1:2; correspond à:
    If (condition) x=1; else x=2;
  - Attente: Sleep(500); // ms
  - Cliquer sur un bouton pour voir les lignes de code correspondantes
  - L'interrupteur « Sorties Actives » doit être enclenché

Variantes possibles : utiliser l'API BC 9020, ou C#, ou Piaget (cf. Annexe i)

Annexe 4

A4. Essais en programmation C++ (2 de 2)



Cf. O. Fuentes et al.,2008

Ecran de l'application standard mise à disposition, avec champs d'entrées et sorties (à gauche). Avec champ supplémentaire « Action » éventuellement ajouté par les étudiants durant la manip. (en haut à gauche), ou complétement modifié (en bas à droite, F. Maiorana et B. Rochat, 2008).

Annexe 5

A5. Commande en Piaget - C# (CS) avec réflexe géré en 1131 au niveau de l'API

- Le Manutec doit se brancher sur l'API Beckhoff BC9020
- (L'état, actif ou non, de Twincat est indifférent pour l'utilisation du BC 9020)
- Charger dans l'API, si nécessaire, (yc Create boot project) le code 1131 du réflexe: « D:25-Code at D\1131 sur BC 9020 yc TwinCAT pour chargement\BC 9020 with PC and Grafcet\BC9020 with Cpp or C#.pr6»;
- Copier le répertoire, faire tourner, puis modifier le code Piaget C# : « D:25-Code at D\Piaget CS with reflex 1131\ … »
  - cliquer sur le fichier … .sln, puis sur le boutin "start" – trangle vert
  - sélectionner le "Beckhoff" (le bouton passe de rouge à magenta)
  - enlever la simulation (le bouton passe de vert à rouge)
  - cliquer la fiche ("form") "Manip 25 BC 9020" en haut à gauche pour l'ouvrir