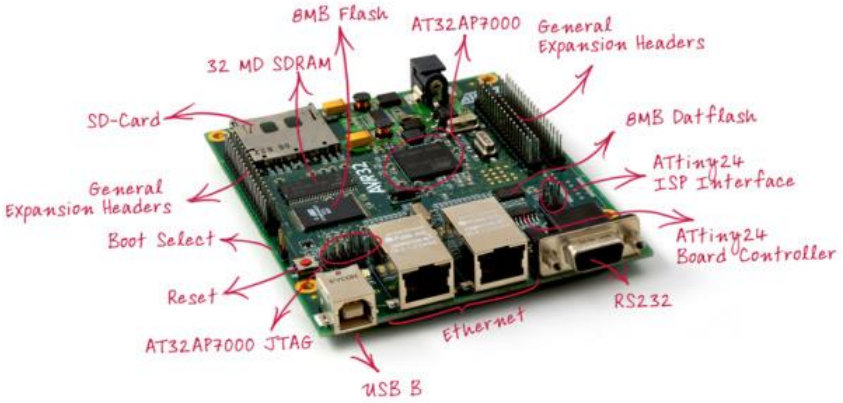
Real-time Systems (TTK4147)
– Ex. 10: Cross development for NGW100
– Miniproject

Bjørn Petter B. Hysing, bjoernh@gmail.com

Norwegian University of Science and Technology

Exercise 10 overview

- AVR32 NGW100 (AT32AP7000)

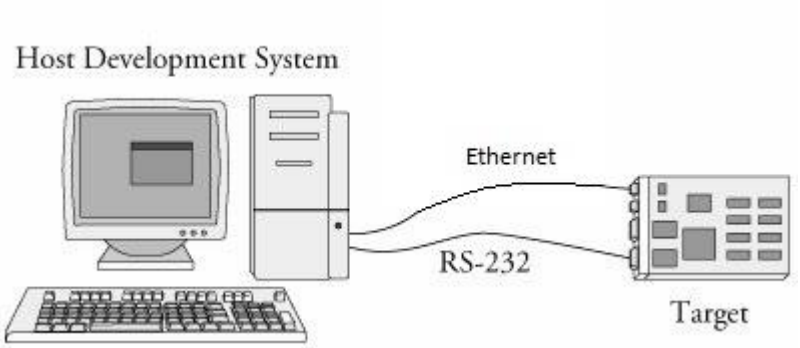


- TASKS:
  - Use Buildroot to create:
    - Linux kernel
    - File system
    - Cross-compiling tools
  - Do some cross development for NGW100
  - Implement a kernel module

- Will use the NGW100 for the miniproject

Cross development

- Software for an embedded system is developed on one platform but runs on another
- The *host system* is the system on which the embedded software is developed. (*Dell computer in lab*)
- The *target system* is the embedded system under development (*NGW100*)
- Reason: An embedded system has little or no user interface and limited computational power

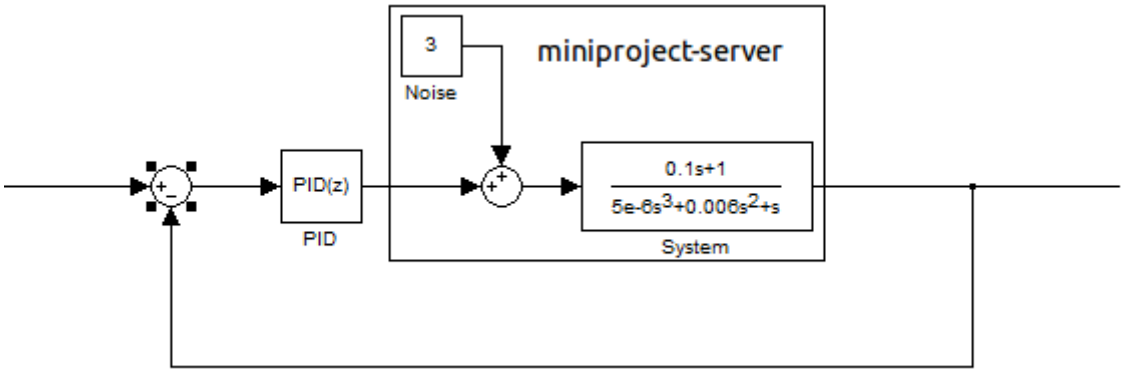


Miniproject – opening remarks

- Deadline Sunday 27th of November (23:59)
- E-mail the results to bjornph@stud.ntnu.no
  - Subject: TTK4147 Miniproject
  - Body: Group member full names
  - Attachment: A zip file with source files (C-files and Makefile only) and plots and explanations (.pdf file)
- Borrowing equipment is possible (However Real-time lab is usually packed Tuesday-Thursday)
- “Studass” will help you with setup and explanation, but do not expect them to tell you how to solve the project in the “best way”.

Miniproject overview

- Control a simulated system with a PI controller
  - miniproject-server (Executable given to you, runs on PC)
  - **miniproject-client:** (You will implement this, runs on NGW100)
- Respond to signals from the server (part 2)



Implementation details

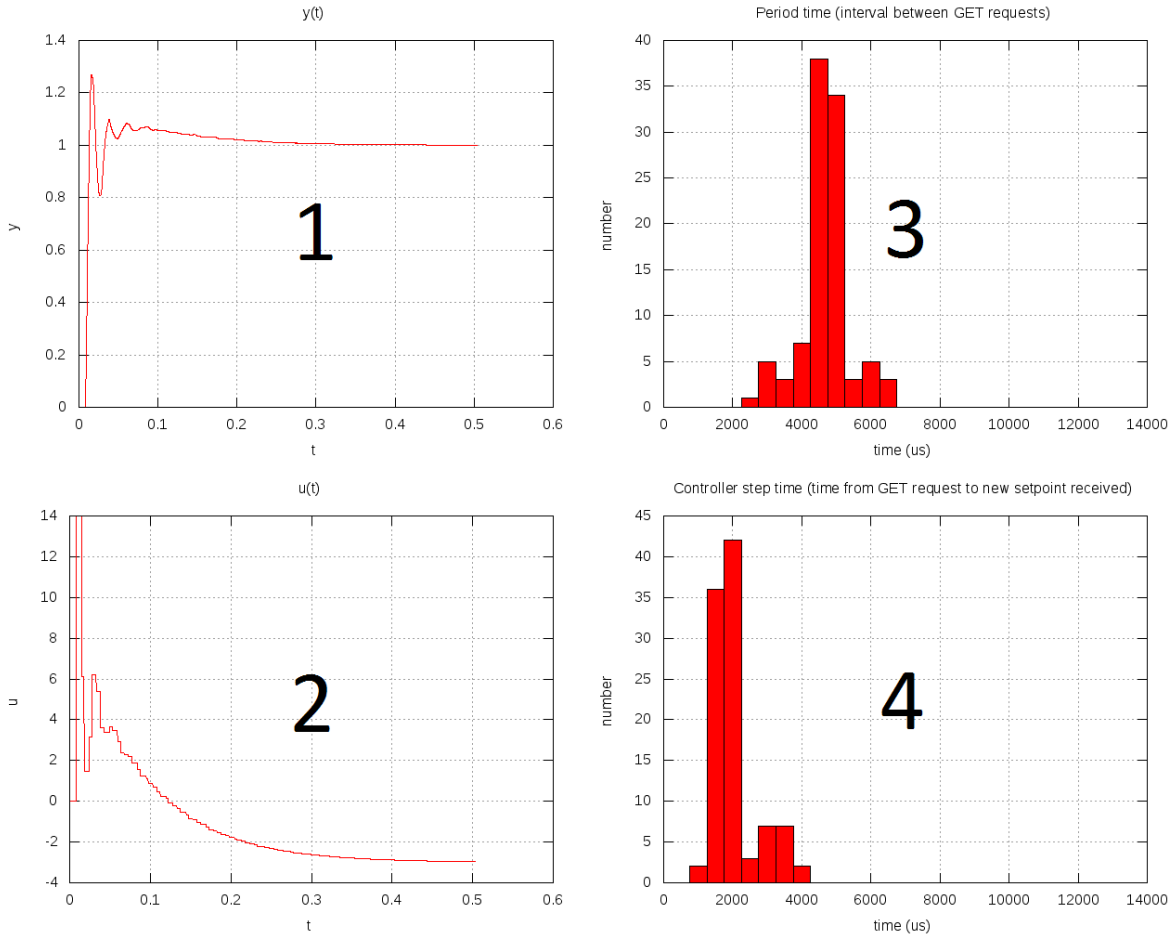
- The implementation should consist of three threads
  - UDP listener
  - A thread implements the PI controller
  - A thread that responds to received signals (part 2)
- Remarks:
  - Code for UDP communication with server will be given
  - Multiple threads are allowed to send UDP packets, but you must make sure that two threads are not trying to send at the same time
  - Some communication/synchronization between your threads will be necessary, how you do solve this is up to you.

Communication commands

Command	String	Direction	Comments
START	“START”	To server	This packet starts the simulation
GET	“GET”	To server	This packet ask for the server to return a packet with y
GET_ACK	“GET_ACK:123.456 ”	From server	This packet is a reply from the server containing the y as a double.
SET	“SET:123.456”	To server	This packet contains the new u value as a double for the simulation
STOP	“STOP”	To server	This packet ends the simulation

Command	String	Direction	Comments
SIGNAL	“SIGNAL”	From server	This packet is a signal that should be responded with a SIGNAL_ACK
SIGNAL ACK	“SIGNAL_ACK”	To server	This is the response to the SIGNAL sent back to the server.

Sample result graph



NTNU

Miniproject tips

- Permission for file (server on Host): `chmod u=rx file`
- Be aware that `printf()` will lead to timing issues

- Do not spend forever to get a “perfect” result:
  - Solve the problem as described in the text
  - Clean your code
  - Explain your solution and though process briefly in a .pdf document
  - Deliver and start reading for exams instead

Check your approved assignment status

- Wrapping up - make sure to get approved all assignments not to lose credits (ask “studass” during lab hours)